VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NEEUKLIDOVSKÝ RAYTRACER

NON-EUCLIDEAN RAYTRACER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN KOSTELNÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ STARKA**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Kostelník Martin**
Program: Informační technologie
Název: **Neeuklidovský raytracer**
Non-Euclidean Raytracer
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte techniku sledování paprsků (ray-tracing) a metody s ní související.
2. Navrhněte interaktivní aplikaci demonstrující nastudované techniky obohacené o práci s neeuclidovským prostorem.
3. Vytvořte navrženou interaktivní aplikaci.
4. Vytvořte krátké video prezentující práci.

Literatura:

- Po dohodě s vedoucím.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, a část bodu 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Starka Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato bakalářská práce se zabývá metodou sledování paprsků v reálném čase obohacenou o práci s neeuklidovským prostorem. Cílem je navrhnout a implementovat interaktivní aplikaci, která umožní uživateli pohyb ve 3D scéně. Veškeré výpočty jsou prováděny na CPU. Jsou implementovány celkem čtyři rozdílné neeuklidovské prvky: portály, zakřivené tunely, škálovací a rotační tunely. Výsledkem práce je interaktivní aplikace obsahující osm ukázkových scén, které demonstrují neeuklidovské prvky.

Abstract

This bachelor's thesis deals with the real-time ray tracing algorithm enhanced by non-euclidean spaces. The goal is to design and implement an interactive application, which allows the user to move around in a 3D scene. All computations are performed by the CPU. In total, four non-euclidean elements are implemented. These include portals, warped tunnels, shrinking and rotation tunnels. The result of this project is an interactive application consisting of eight sample scenes demonstrating the non-euclidean elements.

Klíčová slova

3D, počítačová grafika, vykreslování v reálném čase, ray tracing, obalová tělesa, neeuklidovská geometrie, portály, tunely, C++

Keywords

3D, computer graphics, real-time rendering, ray tracing, bounding volumes, non-euclidean geometry, portals, tunnels, C++

Citace

KOSTELNÍK, Martin. *NEEUKLIDOVSKÝ RAYTRACER*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka

NEEUKLIDOVSKÝ RAYTRACER

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Starky. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Kostelník
11. května 2021

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Tomáši Starkovi za podnětné rady, ochotu a vstřícný přístup při vedení práce. Dále děkuji za podporu své rodině a svým blízkým.

Obsah

Kapitola 1

Úvod

Stále častěji a častěji se můžeme setkat s metodou sledování paprsků (dále jen ray tracing). Jedná se o zobrazovací metodu, která je známá pro kvalitu výsledného obrazu, ale také pro svou výpočetní náročnost. K vykreslení scény je potřeba sledovat jednotlivé paprsky světla, konkrétně jejich kolize s objekty ve scéně, přičemž vznikají nové paprsky simulující odraz světla nebo průhlednost materiálu. To znamená extrémní počet nutných testů kolizí, a proto byla dříve tato metoda vhodná pouze pro statické scény. O rozmach ray tracingu se zasloužila zejména americká společnost Nvidia, která na konvenci Gamescom v roce 2018 oznámila řadu revolučních grafických karet. Oproti svým předchůdcům tyto karty obsahují specializovaný hardware určený k akceleraci výpočtů. Dostali jsme se tak o krok blíže k využití této metody i pro dynamické scény, neboli ray tracingu v reálném čase.

Cílem této práce je navrhnout a implementovat nástroj, s pomocí kterého bude možno vykreslovat scénu metodou sledování paprsků v reálném čase. Scéna bude navíc obsahovat neeuklidovské prvky, jako například portály nebo různě zakřivené části prostoru. Žádaného efektu docílíme manipulací s paprsky při kolizi s neeuklidovským objektem. Můžeme například upravit jeho směr nebo jej přemístit.

Dále bude uživateli umožněno pohybovat kamerou a na vlastní oči vidět netradiční efekty dříve zmíněných neeuklidovských prvků. To přináší další úskalí. Je totiž nutno synchronizovat pohyb kamery se zakřivením prostoru. Způsob synchronizace závisí na konkrétním zakřivení.

S neeuklidovskou geometrií se v reálném světě nesetkáme a ani v jiných aplikacích není vůbec běžná. Použití tohoto nástroje bude tedy vhodné ve velmi exotických případech. Aplikace bude pro veškeré výpočty využívat procesoru počítače. Běžnější praxí je využití grafické karty, nicméně její použití je náročnější a nad rámec této práce.

Základní teorie, pojmy a principy jsou popsány v kapitole 2. Návrh řešení a využité postupy jsou obsaženy v kapitole 3. Kapitola 4 obsahuje použité technologie a popis implementace tohoto nástroje.

Kapitola 2

Teorie

V této kapitole je představena základní teorie, principy a existující řešení související s prací. Blíže je popsán ray tracing a optimalizační metody s ní související. Také jsou rozebrány některé obecné oblasti počítačové grafiky, jako například osvětlovací modely. Závěrečné sekce pojednávají o neeuklidovské geometrii a existujících řešeních.

2.1 Realistické zobrazení 3D scény

Realistické zobrazení je něco, o co se snažíme již od počátků počítačové grafiky. V oboru neexistuje žádný standard, který by pojem realistické zobrazení definoval. Například definice dle doktora Michala Španěla zní: „Realistické zobrazení je proces generování obrazu virtuální počítačové scény tak, aby vizuální vlastnosti a kvalita výsledného zobrazení odpovídaly obrazu (fotografii) příslušné reálné scény.“[1] V praxi to znamená, že pokud se podíváme na dva obrazy stejné scény, jeden reálný a druhý generovaný počítačem, nesmíme je být schopni rozlišit.

Jeden z hlavních přístupů pro generování realistických obrazů je physically based rendering (PBR) [13]. Tento přístup se snaží co nejvěrněji simulovat chování světla. Nejedná se o striktní sadu pravidel či konkrétní implementaci. PBR tak přestavuje spíše sadu principů a technik. Výhodou je, že člověk může navrhnout vlastnosti použitých materiálů na základě jejich fyzikálních vlastností.

Obrázek 2.1: Příklady fotorealistického zobrazení 3D scény.[2]

[1] https://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_slide_raytracing_print.pdf

[2] Přezvato z: https://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_moderni_grafika.pdf

2.2 Ray tracing

První zmínka o vykreslování scény pomocí ray-tracingu se objevila v roce 1968, kdy Arthur Appel publikoval metodu ray casting [1]. Tato metoda nedokázala pracovat se stíny, odrazem a ani lomem světla. Na to v roce 1979 navázal Turner Whitted [16], který ray casting rozšířil o dříve zmíněné prvky. Obohacenou metodu nazval právě ray tracing.

Jedná se o globální vykreslovací metodu, která se snaží simulovat chování světla. Díváme se však na něj obráceně. Nesledujeme paprsky od světelných zdrojů do kamery, ale právě naopak. Paprsky vycházejí z kamery a jsou vrženy do scény, kde kolidují s objekty a odráží se. Tímto způsobem se metoda značně zjednoduší. Pokud by paprsky začínaly ve světelných zdrojích, do kamery by se jich dostal pouze nepatrný zlomek a my bychom tak museli provádět extrémní množství zbytečných výpočtů.

Paprsek

Paprsek je základním kamenem potřebným pro vykreslení scény. Tvoří jej dvě části, počátek (bod v prostoru) a směr (normalizovaný vektor). Matematicky můžeme o paprsku hovořit jako o polopřímce a popisujeme jej parametrickou rovnicí ve tvaru:

$$P(t) = o + t\vec{d},$$

kde t je vzdálenost od počátku směrem ke konci polopřímky. Vzdálenost může nabývat jak kladných, tak i záporných hodnot. Pokud je hodnota záporná, nachází se bod P za počátkem. Pro naše potřeby budeme uvažovat pouze kladné t, jelikož nás zajímají objekty před počátkem.

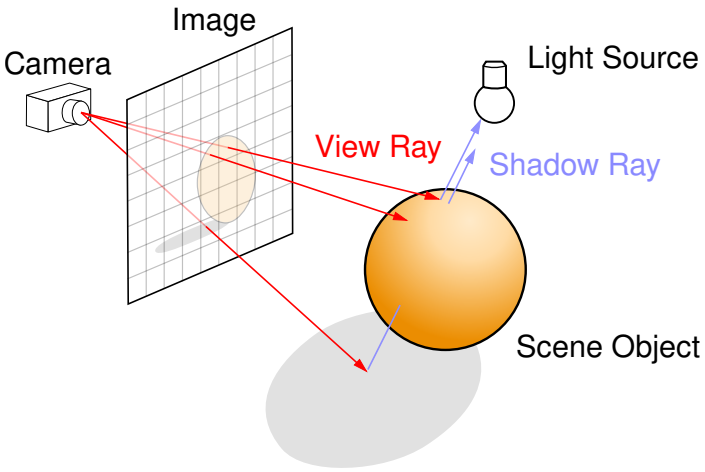
Princip metody

Ray-tracing nabízí velice pohodlný způsob řešení problému viditelnosti. Sledujeme paprsky vržené do scény a jejich interakci s prostředím. Samotný princip metody je tedy prostý, paprsky jsou vrženy do scény z místa pozorovatele skrze průmětnu. Následně je pro každý paprsek nalezen objekt, na který paprsek dopadne. Tento princip je ilustrován na obrázku 2.2.

Typy paprsků

Rozlišujeme následující druhy paprsků:

- Primární paprsek (primary ray, camera ray) - paprsek vržený z kamery. Pro tento paprsek je zejména vypočítán bod dopadu, ze kterého se do scény vysílají ostatní druhy paprsků
- Sekundární paprsek (secondary ray) - vzniká v místě dopadu primárního nebo sekundárního paprsku a simulují odraz či lom světla. Počet sekundárních paprsků může značně převyšovat počet primárních. To záleží na hloubce rekurze.
- Stínový paprsek (shadow ray) - pro každý bod dopadu primárního nebo sekundárního paprsku je třeba vytvořit N stínových paprsků, kde N je počet světelných zdrojů ve scéně. Každý paprsek poté putuje směrem ke korespondujícímu světlu. Výhodou je, že hledáme průsečík na omezenou vzdálenost a navíc nám stačí nalézt jakýkoliv průsečík (ne nejbližší).



Obrázek 2.2: Tento obrázek ilustruje princip metody ray-tracingu, konkrétně vržení paprsků do scény z místa pozorovatele skrze průmětnu, jejich průsečíky s objektem a generování stínových paprsků směrem ke světlu. Převzato z [2].

Průsečík paprsku s trojúhelníkem

Výpočet průsečíku paprsku s trojúhelníkem je velice důležitá operace, která tvoří značnou část výpočetní náročnosti metody. Objekty scény bývají totiž obvykle popsány množinou trojúhelníků, a tak najít průsečík paprsku s objektem znamená najít průsečík paprsku s jedním z trojúhelníků. Jedním z hojně používaných algoritmů pro výpočet je Möller-Trumbore [11], který byl poprvé představen v roce 2005. Princip algoritmu je založen na řešení následující rovnice:

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2,$$

kde levá strana rovnice přestavuje bod nacházející se na paprsku a pravá strana přestavuje bod na trojúhelníku. V_0, V_1 a V_2 jsou vrcholy trojúhelníku a u a v jsou barycentrické souřadnice. Kolize nastala, pokud platí následující:

$$u \geq 0 \wedge v \geq 0 \wedge (u + v) \leq 1$$

2.3 Akcelerační struktury

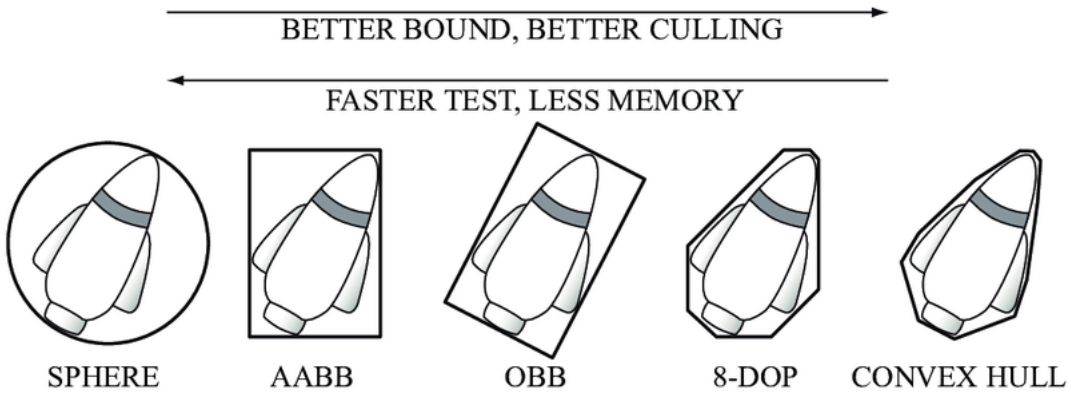
Ray tracing je výpočetně velmi náročná metoda, a proto existuje nekončící snaha o jeho neustálé zrychlování. Většina technik pro urychlení metody spočívá v použití takzvaných akceleračních struktur. Při vykreslování scény strávíme drtivou většinu času počítáním průsečíků paprsků s objekty. Zejména pro trojúhelníky se jedná o časově náročnou operaci, která se v průběhu vykreslování provádí s obrovským počtem opakování. Musíme totiž vypočítat průsečík každého paprsku s každým trojúhelníkem ve scéně. Nemůžeme nijak zkrátit čas potřebný pro výpočet jednoho průsečíku, můžeme ale omezit počet těchto testů.

Akcelerační struktury nám ve výsledku pomáhají v rozhodování, zda paprsek může protnout určité objekty ve scéně a případně tyto objekty vůbec nebrat v potaz jako kandidáty pro kolizi. V této sekci jsou uvedeni tři představitelé akceleračních struktur.

Hierarchie obalových těles

Použití obalových těles je optimalizační metoda, která zjednodušuje ray tracing redukováním počtu nutných výpočtů průsečíků paprsků s primitivy. Spočívá v obalení geometrie scény do určitého tvaru, například kvádru. Různé druhy obalových těles jsou demonstrovány na obrázku 2.3. Při použití složitějšího tvaru dosáhneme těsnějšího obepnutí primitiv, ale na druhou stranu se zvýší složitost detekce kolize s paprskem a také čas konstrukce obalového tělesa. Musíme tedy zhodnotit, zda se nám použití složitějších tvarů vyplatí.

Hlavní výhodou je, že při vykreslování scény nám poté stačí pouze otestovat, zda paprsek protne dané obalové těleso. Pokud ne, nemusíme vůbec testovat kolizi paprsku s primitivy obsaženými v obalu.



Obrázek 2.3: Obrázek ilustruje příklady různých obalových těles i s porovnáním vzhledem k těsnosti obalu, rychlosti konstrukce a rychlosti výpočtu kolize. Na levé straně jsou tělesa, která lze konstruovat velice pohodlně, naopak na pravé straně jsou tělesa s těsným obepnutím. Převzato z [10].

Hierarchie obalových těles [6] je stromová struktura, která seskupuje objekty ve scéně. Samotné objekty jsou zabaleny v obalovém tělese a tyto články tvoří listy stromu. Listy jsou opět seskupeny a společně obsaženy ve větším obalu. Tímto způsobem se postupuje rekurzivně, dokud ve stromě není obsažena veškerá geometrie scény.

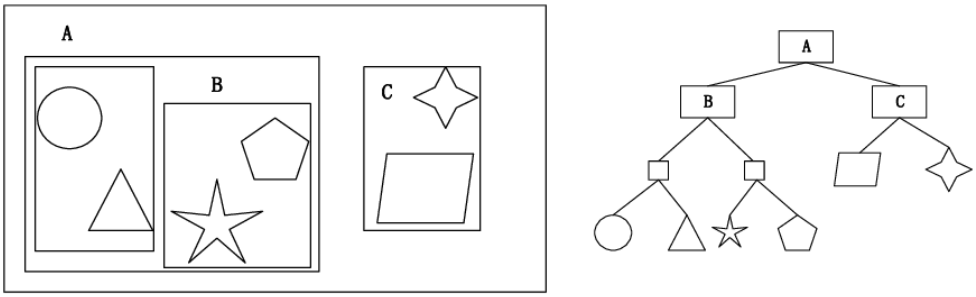
Zjednodušení testů kolizí nastane už při použití jednoduchých obalových těles, nicméně použití hierarchie snižuje i počet nutných testů kolizí s obalovými tělesy. Časová složitost se z lineární redukuje na logaritmickou vzhledem k počtu objektů ve scéně. Toto je způsobeno tím, že nemusíme testovat ty členy stromu, u kterých jsme nemuseli testovat jejich rodiče.

Příklad takové hierarchie i s jednoduchými objekty je demonstrován na obrázku 2.4.

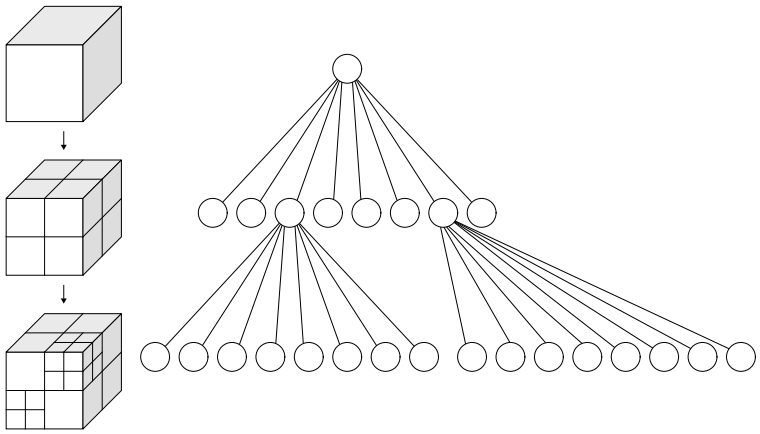
Oktalový strom

Oktalový strom [9] je stromová struktura, která vychází z dělení prostoru na osm částí se stejným objemem. Celou scénu následně představuje kořenový uzel a menší části scény jsou jeho potomci. Z toho můžeme vyvodit, že každý uzel ve stromu má přesně osm potomků. Ukázku rozděleného kusu prostoru a reprezentaci stromem můžeme vidět na obrázku 2.5. Tento přístup může způsobit problémy, pokud geometrie ve scéně není rovnoměrně rozložena. V tomto případě nastane situace, kdy hloubka stromu v určité oblasti je mnohem

větší než v oblastech jiných, což může mít za následek snížení výkonu. Jinými slovy můžeme říci, že adaptabilita struktury na scénu není vždy ideální.



Obrázek 2.4: Příklad vytvořené hierarchie obalových těles. Levá část zobrazuje jednotlivé objekty scény umístěné v obalových tělesech. Pravá část představuje již sestavený strom, kde samotné objekty scény jsou obsaženy v listových uzlech. Převzato z [5].

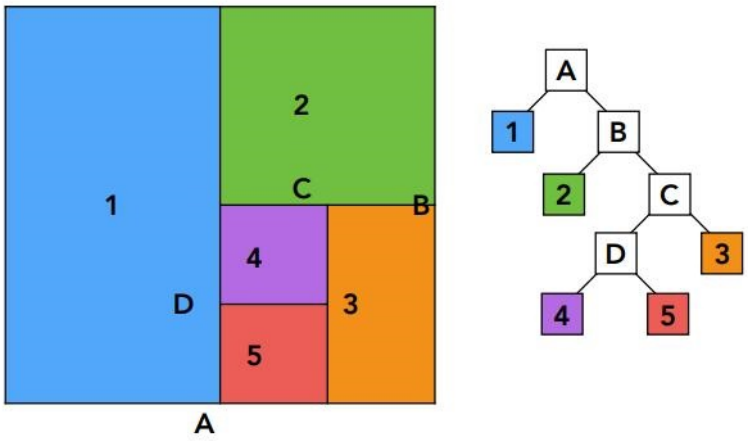


Obrázek 2.5: Obrázek znázorňuje princip dělení prostoru při použití oktalového stromu jako struktury pro akceleraci ray tracingu. Levá část znázorňuje rekurzivní dělení prostoru na osm stejných částí. V pravé části je zobrazen příklad oktalového stromu.[3]

K-d strom

K-dimenzionální stromy [3] jsou další z akceleračních struktur. Fungují na podobném principu jako oktalové stromy, tedy na dělení prostoru. Na rozdíl od nich ale není pozice rozdělení pevně stanovena. Stále se jedná o rozdělení zarovnané s osami, což zjednoduší výpočet průsečíku s oblastí. Důležitým kritériem této struktury je výběr algoritmu pro nalezení optimální pozice a směru, ve kterém má být prostor rozdělen. Základním způsobem je cyklické střídání směru a dělení ve středu oblasti. Existují ale i více sofistikované algoritmy, které určují pozici a směr na základě metriky předpokládané ceny průchodu. Při použití správného algoritmu dosáhneme dobré adaptability struktury na scénu.

[3] Převzato z: https://en.wikipedia.org/wiki/Octree

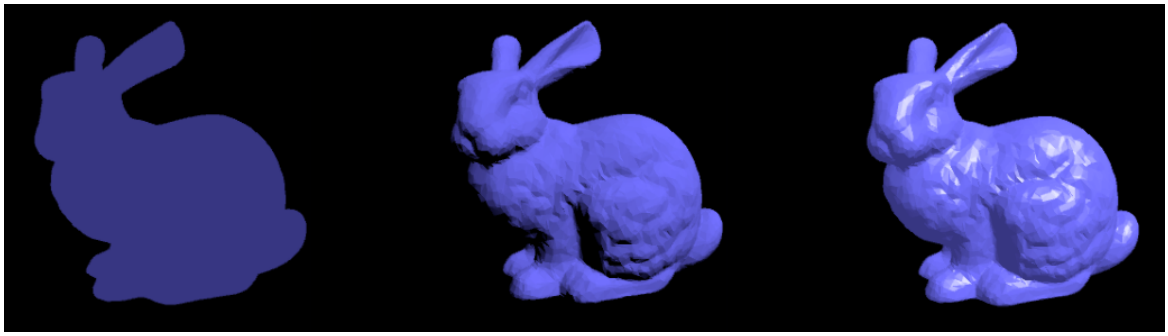


Obrázek 2.6: Princip metody dělení prostoru pro vytvoření K-dimenzionálního stromu ve 2D. Levá část představuje dělení prostoru s cyklickým výběrem směru a dělením v polovině. V pravé části je vytvořený strom.[4]

2.4 Osvětlovací modely

Osvětlovací modely simulují interakci světla s materiálem objektu. V počítačové grafice jsou důležité z důvodu dodání hloubky obrazu. Bez osvětlení by vygenerovaný obraz vypadal ploše, jak je ukázáno na obrázku 2.7. Pomáhají nám také dodat objektu některé specifické efekty, jako zrcadlení nebo průhlednost. V této sekci jsou rozebrány dva velice populární modely, Lambertův osvětlovací model a Phongův osvětlovací model.

Kromě zmíněných existuje i řada dalších osvětlovacích modelů, které interakci světla s materiálem simulují věrněji. Obvykle se ale s rostoucí kvalitou výsledku zvedá i výpočetní náročnost. Typickým příkladem je physically based shading, který nabývá na popularitě jak ve filmovém, tak v herním průmyslu [8].



Obrázek 2.7: Porovnání modelu Stanford Bunny ve třech případech. Na levé straně je vykreslený model bez jakéhokoliv osvětlovacího modelu. Uprostřed je použit Lambertův osvětlovací model a v pravé části Phongův osvětlovací model.[5]

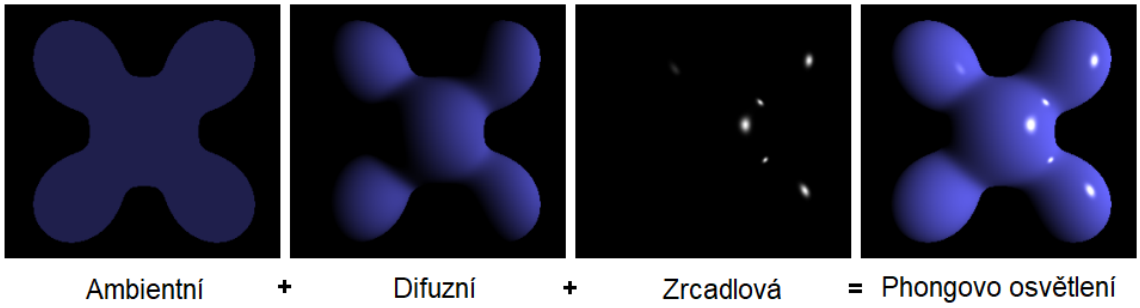
[4] Převzato a upraveno z: https://www.programmersought.com/article/89343606302/
[5] Převzato z: http://viclw17.github.io/2016/04/19/phong-shading-model-walkthrough/

Lambertův osvětlovací model

Jedná se o osvětlovací model pro simulaci difuzních (matných) povrchů. Stal se velice populárním zejména kvůli své jednoduchosti jak implementace, tak i vzhledem k výpočetní náročnosti. K výpočtu barvy vyžaduje pouze pozice světel ve scéně, nikoliv pozici pozorovatele. Nedochází tedy k odrazům světla a na objektu nevznikají odlesky.

Phongův osvětlovací model

Tento osvětlovací model simuluje zejména lesklé povrchy. Poprvé byl navržen a představen Bui Tuong Phongem [14]. Přichází s myšlenkou rozdělit osvětlení do tří složek: difuzní, ambientní a spekulární. Rozdělení je ukázáno na obrázku 2.8. Difuzní složka je vygenerována za pomocí Lambertova osvětlovacího modelu a ambientní složka je dána konstantním množstvím osvětlení v každém bodě scény. Spekulární složka je vypočítána pomocí kosinusu úhlu mezi směrem světla a směrem odrazu světla vzhledem k pozorovateli umocněného na exponent, který ovlivňuje lesklost povrchu. Na rozdíl od Lambertova modelu je Phongův model tedy závislý jak na pozicích světel, tak i na aktuální pozici pozorovatele.



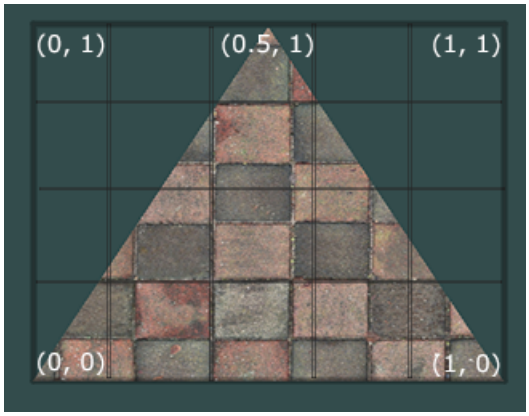
Obrázek 2.8: Obrázek demonstruje jednotlivé komponenty Phongova osvětlení: ambientní, difuzní a spekulární složku. Ambientní složka je konstatní v celé scéně, difuzní je vygenerována Lambertovým osvětlením a spekulární je závislá na pozici pozorovatele.[6]

2.5 Texturování

Texturování je technika, která dodává detail objektům scény. Textura je často 2D obraz, který se za pomoci mapování aplikuje na 3D objekt, čímž vzniká realistický vzhled. Jelikož jsme schopni vložit značnou míru detailu do jednoho obrazu, objekty se jeví mnohem více detailní než ve skutečnosti jsou. Za jiných okolností bychom museli objekt poskládat ze složitější geometrie (přidáním více vrcholů), abychom dosáhli stejného detailu.[7]

Pro namapování textury na trojúhelník musíme ke každému vrcholu přiřadit, ke které části textury náleží. Nalezení konkrétní barvy na povrchu trojúhelníku se následně provádí interpolací. Texturové souřadnice se pohybují v intervalu $\langle 0, 1\rangle$. Souřadnice $[0, 0]$ představuje levý dolní roh obrazu a souřadnice $[1, 1]$ naopak roh pravý horní. Toto je společně s jednoduchou texturou pro trojúhelník vidět na obrázku 2.9.

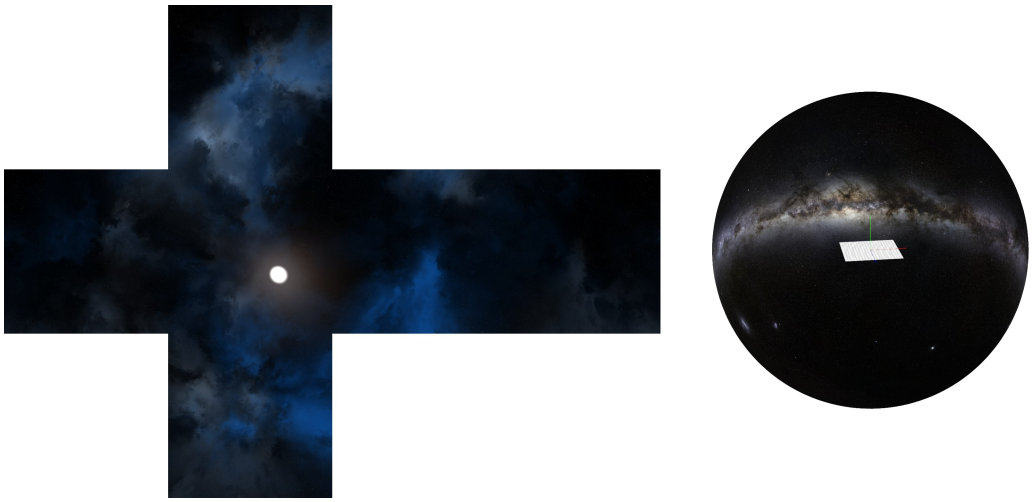
[6] Převzato z: https://en.wikipedia.org/wiki/Phong_reflection_model
[7] https://learnopengl.com/Getting-started/Textures



Obrázek 2.9: Příklad trojúhelníkové textury společně s referenčními souřadnicemi pro mapování na objekt.[7]

2.6 Skybox

Nejčastějším použitím skyboxu je realizace nebes společně s extrémně vzdálenými a nedosažitelnými objekty. Díky tomu je scéna prezentována jako mnohem větší, než ve skutečnosti je. Při použití skyboxu je scéna uzavřena v krychli, na jejíž strany je následně nanesena textura obsahující například mraky, hory, stromy či jakékoliv jiné objekty. Pozorovatel je následně umístěn v samotném středu a krychle se tak jeví v nekonečné vzdálenosti. Scénu je možné obalit i jiným tvarem než krychlí. Velice často se můžeme setkat s kopulí, která je více flexibilní a poskytuje lepší možnosti pro animaci. Na druhou stranu je obtížnější na implementaci a je pomalejší na vykreslení. Příklady takových textur je možné vidět na obrázku 2.10.



Obrázek 2.10: Příklady textur pro realizaci skyboxu. Na levé straně textura pro krychlovitý tvar a na pravé straně textura pro tvar kopule.[8]

[8] Převzato z: https://www.cleanpng.com/free/skybox.html

Při návrhu skyboxu pro určitou scénu je důležité, aby na sebe jednotlivé části dobře navazovaly a nekazily tak dojem skutečného pozadí. Pozorovatel nesmí být schopen rozpoznat, že se ve skutečnosti nachází uprostřed krychle.

2.7 Neeuklidovská geometrie

V této sekci je rozebrána neeuklidovská geometrie ([12], [7], [4]) jak z matematického hlediska, tak i z hlediska obecné mluvy. Neeuklidovská geometrie v matematickém smyslu má přesný popis a je založená na postulátech formulovaných Euklidem, ke kterým jsou přidány další. Pokud matematickou definici pomineme, můžeme za neeuklidovskou geometrii považovat jakékoliv prvky prostoru, které nějakým způsobem mění jeho topologii (např. portály).

Vznik neeuklidovské geometrie se datuje do starověkého Řecka, kdy se Euklides snažil shromáždit všechny poznatky o geometrii. Jeho cílem bylo také dokázat každé tvrzení pomocí jednodušších. Při svém bádání vytvořil systém pěti postulátů, ze kterých lze odvodit veškeré znalosti o geometrii. Nebyl však spokojen s pátým postulátem a věřil, že ho lze odvodit ze čtyřech předcházejících. Toto odstartovalo závod, kde se mnozí matematici snažili o odvození pátého postulátu, ale žádnému se to nepovedlo.

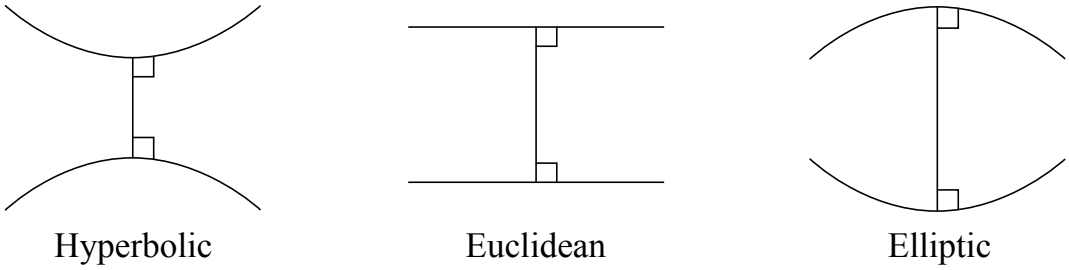
Přelom v tomto odvětví nastal až na začátku 19. století, kdy nezávisle na sobě ruský matematik Nikolaj Ivanovič Lobačevskij a maďarský matematik János Bolyai představili svět, ve kterém platí první čtyři postuláty, ale nikoliv pátý, což dokazuje, že jej nelze žádným způsobem odvodit. Neeuklidovská geometrie získala svůj název právě proto, že samotný Euklid nevěřil v její existenci.

Neeuklidovská geometrie v matematice

Abychom si mohli zadefinovat pojem neeuklidovská geometrie, musíme se podívat na geometrii absolutní. Absolutní geometrie je axiomatický systém, který je tvořen prvními čtyřmi Euklidovými postuláty a všemi větami o geometrii, které se dají dokázat bez použití pátého postulátu. Neeuklidovská geometrie se potom skládá z absolutní geometrie a jednoho přidaného postulátu, který nahradí původní pátý Euklidův postulát. Důležitými zástupci těchto geometrií jsou hyperbolická a eliptická geometrie.
Euklidovy postuláty:

1. Dva body určují jedinou úsečku, která v těch bodech končí.
2. Každá úsečka může být prodloužena tak, že vznikne nová úsečka.
3. Je možné nakreslit kružnici s libovolným středem a poloměrem.
4. Všechny pravé úhly jsou si rovny.
5. Mějme přímku a bod, který na dané přímce neleží. Tímto bodem lze vést nanejvýš jednu rovnoběžku s danou přímkou.

Jedním z hlavním rozdílů euklidovské a neeuklidovské geometrie je chování rovnoběžek. Různé chování je demonstrováno na obrázku 2.11, ze kterého je patrné, jak hyperbolická a eliptická geometrie porušuje pátý Euklidův postulát. V hyperbolické geometrii platí první čtyři postuláty a pátý je nahrazen následujícím.



Obrázek 2.11: Obrázek demonstruje různé chování rovnoběžek v neeuklidovské geometrii. Je znázorněno různé chování v prostoru hyperbolickém (vlevo), euklidovském (uprostřed) a eliptickém (vpravo).[9]

- V rovině procházejí bodem mimo přímku alespoň dvě různé s ní se neprotínající přímky.

V eliptické geometrii dostal nový pátý postulát název Lobačevského axiom.

- V rovině neexistuje k dané přímce přímka, která ji neprotíná.

Neeuklidovská geometrie ve filmovém a herním průmyslu

Pokud pomineme striktní matematický význam slovního spojení neeuklidovská geometrie, můžeme mluvit o jakýchkoliv objektech, které nějakým způsobem mění topologii prostoru. Takové objekty jsou tématem zájmu ve filmovém průmyslu v žánru science fiction, kde jsou vysvětleny fyzikálními jevy, nebo v žánru fantasy, kde se často používá pojem magie. Ukázky z filmových děl, které využily neeuklidovské prvky jsou na obrázku 2.12.

Mluvíme například o teleportačních zařízeních, které určitým způsobem spojují dvě nebo více částí prostoru. Konkrétními příklady různých jevů a existujícími řešeními se blíže zabývá sekce 2.8.

Obrázek 2.12: Obrázek demonstruje použití neeuklidovských prvků ve filmovém průmyslu, konkrétně ve filmu Dr. Strange od studia Marvel Entertainment. V levé části je vidět portál spojující dvě části prostoru použitý jako dálkové komunikační zařízení. Na pravé části je příklad zakřiveného prostoru.[10]

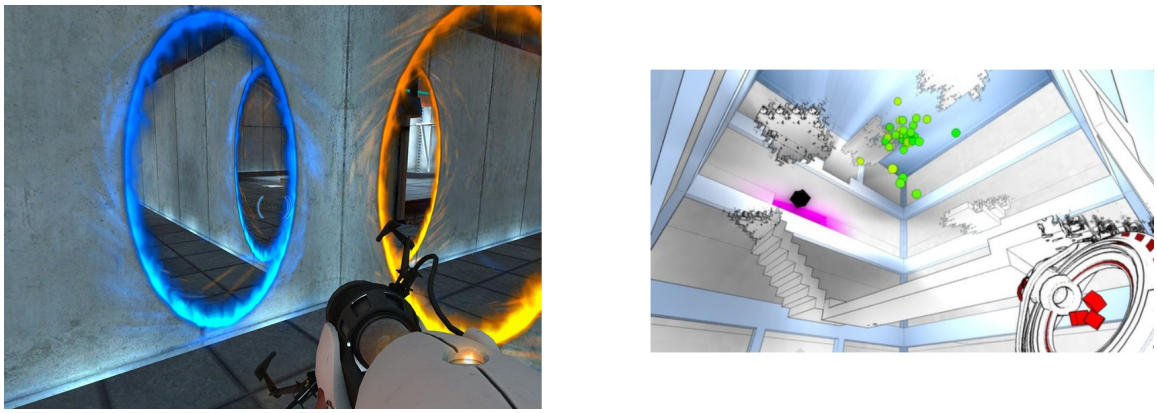
[9] Převzato z: https://commons.wikimedia.org/wiki/File:Noneuclid.svg
[10] Převzato z https://marvelcinematicuniverse.fandom.com/wiki/Doctor_Strange_(film)/Gallery

2.8 Existující řešení

Neeuklidovské prvky se v dnešní době staly součástí mnoha logických počítačových her. Mezi nejznámější z nich určitě patří hra Portal 2[11], která staví své herní mechaniky na využití portálů k řešení různých logických problémů. Portály umožňují nejen přemístění hráče, ale i průchod laserových paprsků, mostů a obecně jakýchkoliv prvků scény. Hra nabízí jak kampaň pro jednoho hráče, tak i kooperační mód, kde se dva hráči mohou snažit překonávat levely vytvořené komunitou.

Druhá počítačová hra stojící za zmínku, je Antichamber[12]. Jedná se o logickou hru z pohledu první osoby, kde hráč prochází několik levelů plných logických úloh a hádanek, ve kterých musí využívat objekty a prvky neexistující v reálném světě. Kromě toho má hra i psychologickou stránku, ukazuje totiž hráči řadu přísloví, které mu mají pomoci hádanky vyřešit. Vše je zakorporováno a designováno do stylu M. C. Eschera, který měl v oblibě ztvárňování perspektivní paradoxy a jiné netradiční topologické útvary.

Tea For God[13] je další ze zajímavých implementací. Jedná se o akční hru ve virtuální realitě s podporou zařízení Oculus Rift a HTC Vive, která využívá nereálných topologií a procedurálního generování[14].

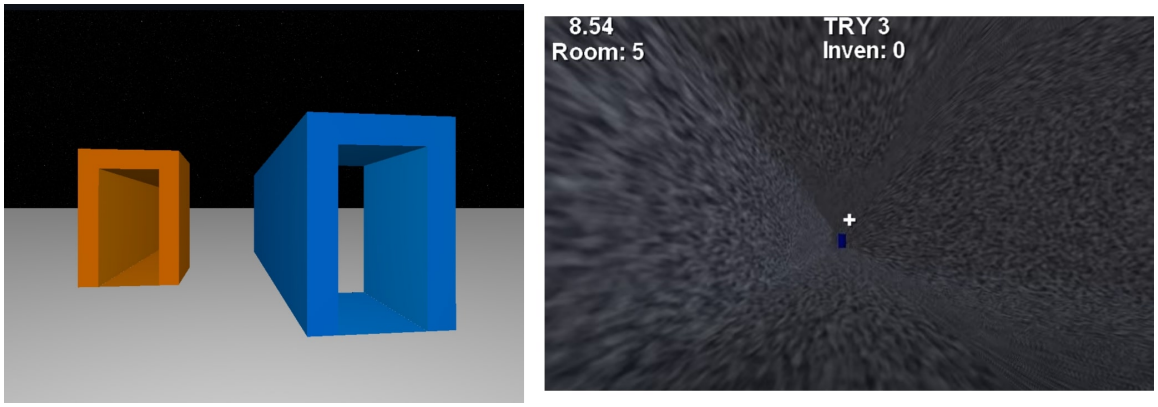


Obrázek 2.13: Na levém obrázku lze vidět ukázku portálů z počítačové hry Portal 2 a na pravém snímek ze hry Antichamber.

Kromě počítačových her našla neeuklidovská geometrie své místo také ve studentských, či volnočasových projektech, které se liší v přístupu k implementaci, použitých metodách a navrhnutých scénách. Programátoři se snaží vytvořit originální scény s unikátními prvky, ať už za účelem sebevzdělávání nebo jako návrh herních mechanik.

Jedním z těchto projektů je NoEuclid[15], neeuklidovský ray tracing engine implementován na GPU. Zdrojový kód je volně dostupný v repozitáři na stránce GitHub. Ukázku můžeme vidět na obrázku 2.14, kde se kamera nachází uprostřed tunelu, který je v pravé části zakřiven. Součástí projektů jsou rozsáhlé scény využívající více druhů iluzí a zakřivení. Autoři také publikovali video[16] demonstrující projekt.

[11] https://store.steampowered.com/app/620/Portal_2/
[12] https://store.steampowered.com/app/219890/Antichamber/
[13] https://void-room.itch.io/tea-for-god
[14] Procedurální generování je programovací technika pro algoritmické vytváření obsahu.
[15] https://github.com/cnlohr/noeuclid
[16] https://youtu.be/tl40xidKF-4



Obrázek 2.14: Ukázka zakřivených tunelů z projektu Euclider a ukázka částečného zakřivení z projektu NoEuclid.

Dalším projektem je Euclider[17], prototyp neeuklidovského herního enginu vytvořeného v programovacím jazyce Rust. Vykreslování scény probíhá na CPU počítače namísto grafické karty. Kombinuje části prostoru vytvořené na základě matematické definice neeuklidovské geometrie s jednoduššími optickými iluzemi, které jsou k vidění na obrázku 2.14.

[17] https://github.com/Limeth/euclider

Kapitola 3

Návrh řešení

Kapitola popisuje vlastní návrh řešení daného problému. Konkrétně je rozebráno upřesnění cílů a očekávaných výsledků projektu. Dále jsou představeny některá důležitá rozhodnutí, která z velké míry ovlivňují implementaci nástroje. Značnou částí kapitoly je i návrh neeuklidovských prvků, kde jsou popsány prvky vybrané k implementaci. Poslední sekce se zabývá návrhem aplikace jako funkčního celku.

3.1 Upřesnění cílů

Cílem práce je vytvořit interaktivní aplikaci, kde jádrem je vykreslování 3D scény za pomoci ray-tracingu v reálném čase. Samotná metoda je blíže popsána a zpracována v sekci 2.2. Aplikace bude také uživateli umožňovat manipulaci s kamerou umístěnou ve scéně. Pohyb kamery bude umožněn prostřednictvím klávesnice počítače a ovládání bude inspirováno počítačovými hrami s pohledem z první osoby.

Dalším cílem je navrhnout a implementovat řadu neeuklidovských prvků, které určitým způsobem mění topologii prostoru. Prvky jsou inspirovány existujícími řešeními popsanými v sekci 2.8. Součástí práce jsou i prvky nové, nepředstavené ve zmíněné sekci. Jejich návrh, společně s ostatními, je popsán v sekci 3.3. Tyto prvky budou následně součástí několika jednoduchých scén, které mají za úkol prvky demonstrovat a nabídnout tak uživateli přímý pohled na jimi způsobené optické iluze. Mezi těmito scénami bude možné libovolně přepínat za běhu programu. Návrh tohoto principu je blíže popsán v sekci 3.4.

3.2 Využití CPU vs. GPU při vykreslování

Jedním z prvních úkolů při návrhu nástroje bylo zhodnotit a rozhodnout, zda bude scéna vykreslována za použití grafické karty nebo procesoru počítače. Tradičnějším přístupem řešení problému je použití GPU, ale obě varianty přinášejí své výhody a nevýhody, které si dále rozebereme.

Implementace na GPU je náročnější a vyžaduje použití vybraného grafického API, se kterým se programátor musí seznámit a naučit pracovat. Tento přístup přináší další úskalí při ladění programu[1]. Neexistuje totiž žádná zpětná vazba ve formě výpisu do konzole nebo možnost použití záchytných bodů v kódu. Ladění je také závislé na použitém

[1] https://learnopengl.com/In-Practice/Debugging

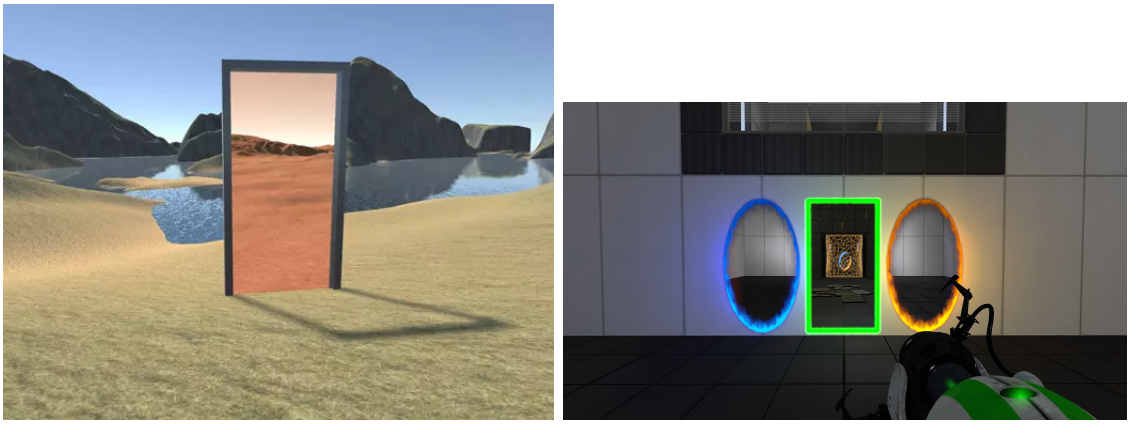
API a konkrétní grafické kartě[2]. Hlavní výhodou naopak je, že grafická karta je hardware speciálně navržený právě pro vykreslování. Při správném použití můžeme očekávat značný nárůst ve výkonu aplikace, zejména v dnešní době, kdy se začínají objevovat grafické karty s podporou ray-tracingu v reálném čase.

Implementace na hlavním procesoru počítače je snadno představitelná. Programátor se nemusí učit novým výpočetním architekturám a pro implementaci mu postačí znalost vybraného programovacího jazyka. Navíc bude mít plnou podporu ladících nástrojů a profilerů. Za toto pohodlí musí ale zaplatit daň v podobě menšího výkonu a případné nutnosti paralelizovat výpočty manuálně.

Po domluvě s vedoucím práce a vzhledem k minimálním zkušenostem autora práce s použitím grafických API a tvorbou aplikací využívající GPU bylo rozhnoduto, že nástroj bude využívat pouze hlavní procesor počítače. To umožní soustředit se více na správnou implementaci neeuklidovských prvků a následnou synchronizaci pohybu kamery se změněnou topologií prostoru.

3.3 Návrh neeuklidovských prvků

Zde se podíváme na návrh neeuklidovských prvků, které mají za úkol vytvořit v aplikaci řadu optických iluzí a zvláštních konstrukcí změnou topologie prostoru v určité oblasti. Jsou navrhnuty celkem čtyři rozdílné prvky. První z nich jsou portály, které umožňují například tvorbu komplexních herních levelů nebo logických hádanek. Dále je navrhnuta řada tunelů s rozdílnými vlastnostmi. Návrh každého prvku i s ilustračními obrázky je dále rozebrán níže v této sekci.



Obrázek 3.1: Inspirace pro návrh portálů do nástroje pro neeuklidovský ray-tracing. Na levém obrázku je obdélníkový portál vestavěný do pevných okrajů vedoucí na jinou planetu.[3]Pravý obrázek ukazuje portály z počítačové hry Portal 2. Ty jsou vestavěny do již existujících stěn a jsou mezi sebou navzájem propojeny.[4]

[2]Pro grafické karty značky NVIDIA existuje nástroj NVIDIA Nsight, který slouží jako plugin do některých populárních vývojových prostředí. Více na https://developer.nvidia.com/nsight-visual-studio-edition

[3]Převzato z: https://assetstore.unity.com/packages/tools/particles-effects/seamless-portals-easy-to-use-portals-with-seamless-transitions-96404

[4]Převzato z: https://www.zing.cz/novinky/96265941/mod-pridal-do-portal-2-cestovani-casem/

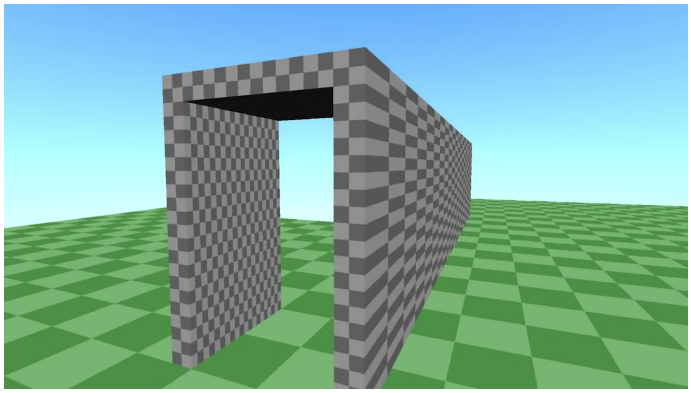
Portály

Portály jsou fenomén, který mnoho lidí zaujme na první pohled. Je tedy logické je v tomto nástroji zahrnout. Portál je 2D nebo 3D objekt, který dokáže přemístit hmotu z bodu A do bodu B. V mnoha implementacích se objevují jak portály jednosměrné, tak i obousměrné. Pro tento projekt jsou portály navrženy jako jednosměrné, v praxi to bude znamenat, že se uživatel bude moci teleportovat tam, ale už ne zpět. Existující implmenetace portálů, které byly použity jako inspirace pro projekt, jsou na obrázku 3.1.

Portály mohou být využity samostatně, buď jako prostředek pro překonávání velkých vzdáleností, nebo tvorbu logických hádanek. Zajímavých efektů dosáhneme i kombinací dvou a více takových portálů najednou. Pokud například postavíme dva portály shodné velikosti naproti sobě, vytvoříme efekt nekonečně dlouhého prostoru, podobně jako při použití dvou zrcadel.

Zakřivené tunely

Druhým neeuklidovským prvkem jsou zakřivené tunely, které budou simulovat stlačení a roztažení prostoru uvnitř, což vytvoří netradiční optickou iluzi. V prvním případě, tedy u stlačeného prostoru, se bude tunel jevit kratším z vnitřní strany a delším z vnější strany. Cesta skrz tento tunel tedy bude rychlejší než cesta okolo. V případě roztaženého prostoru se naopak bude tunel jevit kratší z venku a delší uvnitř. Zde bude cesta skrz tunel pomalejší než cesta kolem. Příklad takového tunelu můžeme vidět na obrázku 3.2, který byl inspirací pro tvorbu tohoto prvku.



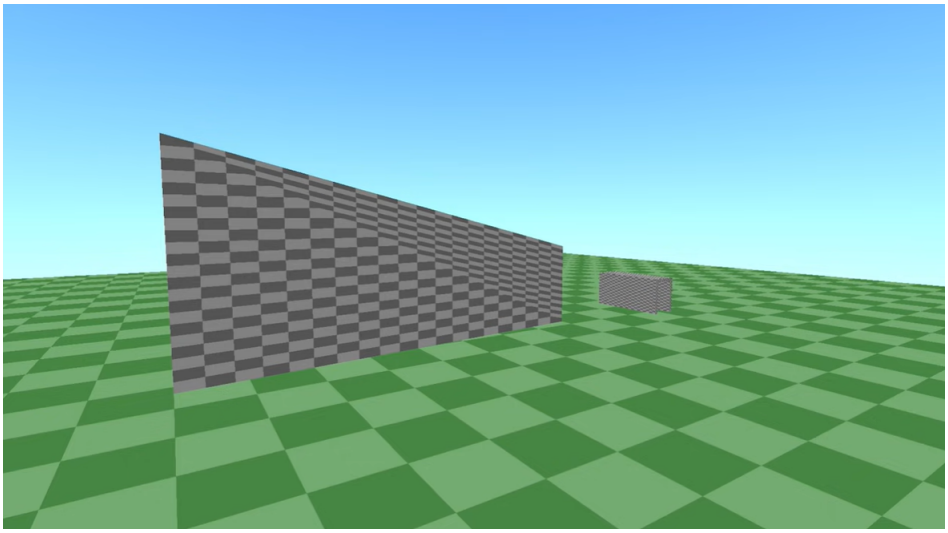
Obrázek 3.2: Inspirace pro návrh zakřiveného tunelu do nástroje pro neeuklidovský ray-tracing. Prostor uvnitř tunelu je stlačen, což vytváří iluzi, že tunel je uvnitř kratší než z vnější strany. Cesta skrz tunel je navíc rychlejší než cesta okolo.[5]

Škálovací tunel

Dalším prvkem bude škálovací tunel, který bude na základě uražené vzdálenosti ve směru tunelu buď zmenšovat, nebo zvětšovat velikost uživatele. Tunel bude fungovat v obou směrech s opačným efektem. To znamená, že pokud uživatel bude procházet jedním směrem, bude postupně zmenšován. Pokud následně projde tunel směrem opačným, bude zpět zvětšen na původní velikost. Zmenšení a zvětšení bude probíhat na základě procentuální změny.

[5] Převzato z: https://youtu.be/kEB11PQ9Eo8

Nemůže se tedy stát, že by uživatel dosáhl nulové velikosti. Dalším důsledkem je, že zvětšování nebude probíhat lineárně, což umožní uživateli dosáhnout enormní velikosti. Jedinou limitací v tomto ohledu bude výška tunelu. Může nastat situace, že uživatel dosáhne takové velikosti, že nebude schopen znovu tunelem projít. Princip tunelu je ilustrován na obrázku 3.3.



Obrázek 3.3: Inspirace pro návrh škálovacího tunelu do nástroje pro neeuklidovský ray-tracing. Na pravé straně lze vidět tunel, který je příliš malý pro průchod uživatele. Větší tunel na levé části však při průchodu změní velikost postavy a umožní tak průchod tunelem menším.[5]

Rotační tunel

Posledním neeuklidovským prvkem je rotační tunel. Jak již z názvu vyplývá, průchod tímto tunelem bude otáčet kamerou kolem předem určené osy v souřadnicovém systému kamery. S pomocí takového tunelu může být dosáhnuto zajímavých herních prvků, uživatel se může pohybovat kamerou vpřed, ale přitom opsat kružnici nebo být otočen hlavou vzhůru.

3.4 Návrh interaktivní aplikace

Již od počátků implementace se objevovaly problémy s optimalizací. I u jednoduchých scén, které obsahovaly řádově desítky primitiv, bylo dosaženo rychlosti vykreslování pod jeden snímek za sekundu. U složitějších modelů se stovkami primitiv trvalo vykreslení jediného snímku několik desítek vteřin, někdy až několik minut. V tu chvíli bylo rozhodnuto, že aplikace bude koncipována do více extrémně minimalistických scén. Každá z těchto scén pak bude demonstrovat jeden druh neeuklidovského prvku.

Pro akceleraci ray tracingu byla zvolena jednoduchá obalová tělesa. Pokud scéna obsahuje malé množství objektů, dosáhneme i tak značného nárůstu výkonu. Ukázkové scény jsou extrémně minimalistické a každá obsahuje řádově jednotky objektů. Při použití hierarchie obalových těles dojde k nárůstu počtu těles ve scéně a společně s režií průchodu by její použití mohlo mít spíše negativní dopad. Hlavní akcelerace je tedy zajištěna kombinací paralelizace výpočtů a použití obalových těles.

Kapitola 4

Implementace řešení

Kapitola se zaměřuje na vlastní popis implementace, včetně všech částí výsledné aplikace. Popisuje implementační detaily a postupy použité při vykreslování a inkorporaci neeuklidovských prvků do systému. Použité technologie, vývojářské nástroje a knihovny, včetně jejich verzí, jsou popsány v sekci 4.1.

4.1 Použité technologie

Aplikace je implementována v programovacím jazyce C++, vzhledem k tomu je třeba vybrat vhodnou knihovnu umožňující vytváření oken, zpracování vstupů z periferií a zachytávání událostí. Pro sestavení projektu je využit systém CMake, jehož hlavní výhodou v tomto projektu je snadné přidání potřebých knihoven. Další výhoda CMake je nezávislost na cílové platformě, to je ale pro tento projekt nepodstatné z důvodu zacílení na jedinou platformu, kterou je operační systém Windows 10 a překladač MSVC. Pro správu verzí je použit systém Git a zdrojové kódy jsou volně dostupné v repozitáři na webové službě GitHub[1]. K projektu je přiložena dokumentace vygenerovaná nástrojem Doxygen.

Použité knihovny

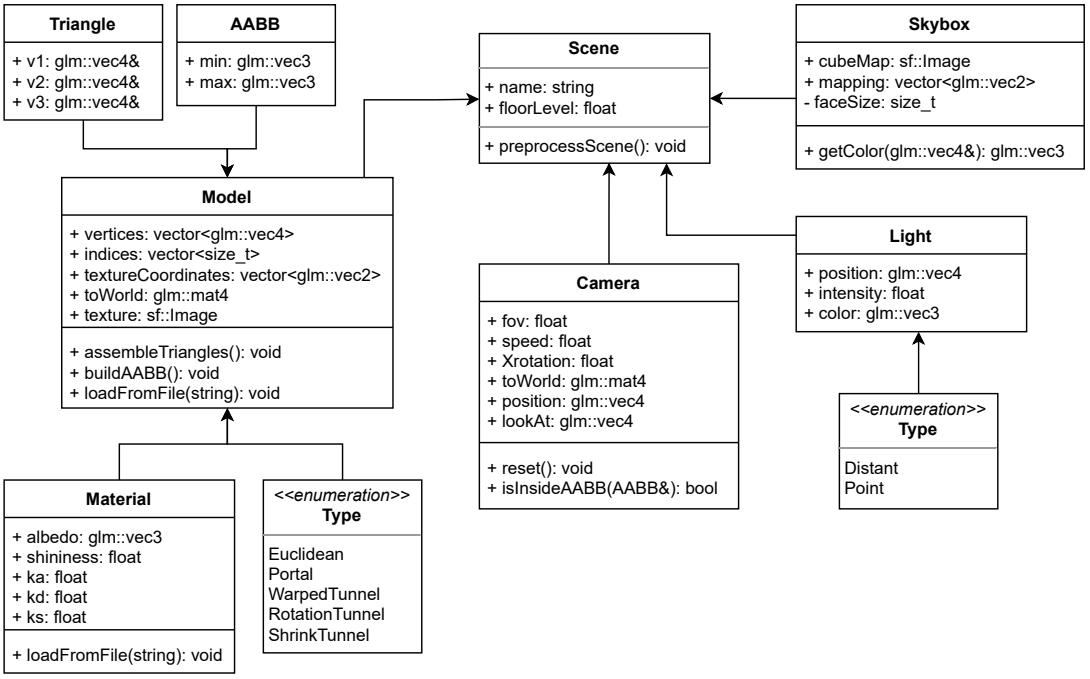
- SFML[2] (Simple and Fast Multimedia Library) je platformově nezávislá knihovna pro vývoj multimediálních aplikací a počítačových her. Poskytuje jednoduché rozhraní ke komunikaci aplikace s několika počítačovými komponenty. Knihovna je složena z celkem pěti modulů, nicméně pro tento projekt jsou potřebné pouze moduly `system`, `graphics` a `window`. Projekt očekává verzi knihovny SFML 2.5.1. Knihovna je využita zejména pro vytvoření a správu hlavního okna aplikace a samotné zobrazení vykresleného snímku na obrazovku počítače. Dalšími úkony jsou zpracování vstupních signálů z klávesnice a myši, uložení a správa textur a v neposlední řadě měření času mezi snímky, který je použit jak pro zobrazení počtu snímků za sekundu, tak pro výpočet vzdálenosti pohybu kamery.
- GLM[3] (OpenGL Mathematics) je matematická knihovna pro jazyk C++ založená na specifikacích jazyka GLSL. Knihovna podporuje akceleraci výpočtů v podobě vek-

[1] Zdrojové kódy nástroje jsou dostupné na: `www.github.com/FazOlek/NoEucRT`
[2] Webové stránky SFML: `https://www.sfml-dev.org/`
[3] Hlavní repozitář knihovny GLM: `https://github.com/g-truc/glm`

torizace za pomocí technologie SIMD (Single instruction, multiple data). Je očekáváno použití verze 0.9.9.8.

- OpenMP[4] je sada direktiv pro kompilátory, které umožňují programátorům jednoduše implementovat vícevláknové aplikace. Do projektu je OpenMP připojeno použitím přepínače `/openmp:experimental` u kompilátoru MSVC. Použití OpenMP je pro projekt klíčové, zajišťuje totiž paralelizaci ray-tracingu a bez něj výkon aplikace výrazně klesne.

4.2 Reprezentace scény



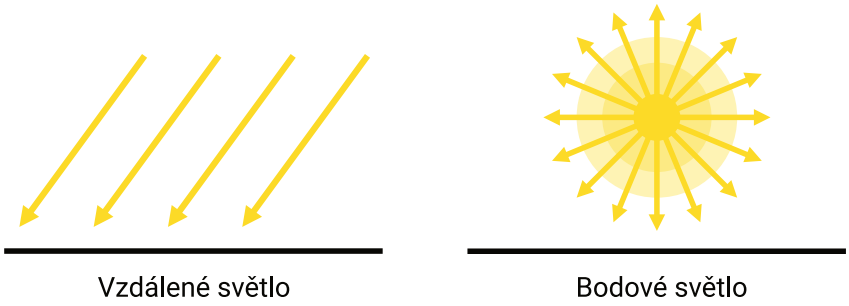
Obrázek 4.1: Diagram tříd demonstrující třídy související s reprezentací objektů ve scéně.

Jedním z prvních problémů, které bylo nutné vyřešit je, jak budou jednotlivé objekty scény reprezentovány v rámci programu. Bylo nutné vymyslet způsob, jak vhodně popsat světelné zdroje, kameru a fyzické objekty, které se dále dělí na euklidovské a neeuklidovské. Každý objekt musí obsahovat transformační matici, s jejíž pomocí získáme jeho pozici v souřadnicovém systému celé scény. Prvotní návrh pro tuto myšlenku byl vytvořit abstraktní třídu, která by zastřešovala libovolný objekt scény a konkrétní typy by zdědily její vlastnosti. Tento přístup se nakonec ukázal nepraktický, zejména kvůli velkým rozdílům mezi jednotlivými druhy objektů a složitosti návrhu. Jelikož první verze programu neobsahovala žádné neeuklidovské prvky, byly vytvořeny tři nezávislé třídy pro světla, kameru a modely, ke kterým byly v pozdější fázi vývoje přidány další třídy pro reprezentaci neeuklidovských prvků. Výsledný třídní model je k vidění na obrázku 4.1.

V programu jsou implementovány dva druhy světel: distanční a bodová. Distanční světlo je reprezentováno pouze směrem, jeho pozice nehraje roli. Bodová světla jsou naopak určena pozicí v prostoru přímo v souřadnicovém systému scény. Ani pro jeden druh světel

[4]Webové stránky OpenMP: https://www.openmp.org/

tak nemusíme ukládat transformační matice, ani spotřebovat výkon na jejich transformace. Barva světla je ve formátu RGB, kde každý komponent je v intervalu $\langle 0, 1 \rangle$. Dalším parametrem je intenzita světla. To může být libovolné kladné desetinné číslo. U distančních světel je vhodné zvolit intenzitu v intervalu $\langle 0, 255 \rangle$. Intenzita bodového světla představuje, jak dobře se světlo šíří prostorem. Její hodnota bývá mnohem větší než u světel distančních. Oba druhy světel znázorňuje obrázek 4.2.



Obrázek 4.2: Obrázek ilustruje princip dvou typů implementovaných světel v aplikaci. Na levé straně je distanční světlo, které má intenzitu nezávislou na vzdálenosti. Na pravé straně je světlo bodové, u kterého se naopak intenzita snižuje s čtvercem vzdálenosti.

Pozice a orientace kamery je reprezentována třemi parametry. Nejdůležitějším je transformační matice do souřadnicového systému scény. Dále pak pro zjednodušení některých výpočtů je uložena její aktuální pozice a směrový vektor, který udává směr pohledu kamery. Ostatními parametry jsou zorné pole ve stupních, rychlost kamery za sekundu při pohybu a pro implementaci některých jevů bylo nutné přidat i aktuální otočení kolem osy x.

Kamera se skládá zejména z transformační matice do souřadnicového systému scény, úhlu zorného pole ve stupních a rychlostí pohybu kamery. Dále pak pro zjednodušení některých výpočtů je uložena její aktuální pozice a směrový vektor, který udává směr pohledu kamery. Pro implementaci některých jevů bylo nutné uložit aktuální otočení kolem osy x.

Fyzické modely jsou popsány pomocí vrcholů s využitím indexace pro snížení paměťových nároků. Každá trojice indexů ve výsledku popisuje jeden trojúhelník. Pro implementaci texturování obsahuje každý model obrázek textury, který je uložen s využitím datové struktury `sf::Image` knihovny SFML, a seznam texturovacích souřadnic, které jsou při načítání namapovány na příslušné vrcholy. K podpoře osvětlování byla zvlášť vytvořena třída pro popis vlastností materiálu objektu. Jako poslední stojí za zmínku typ objektu, který je buď euklidovského, nebo jedním z neeuklidovských charakterů.

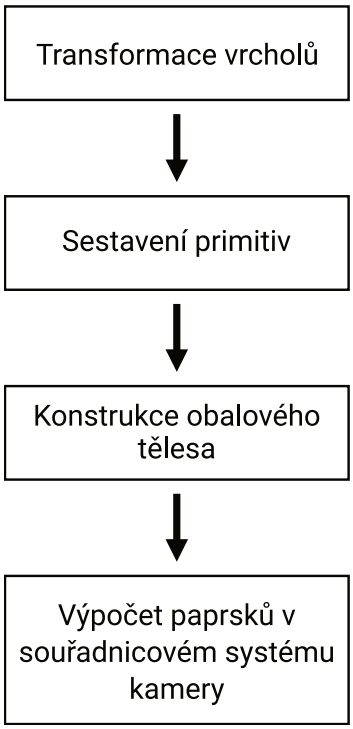
Načítání objektů

Pro pohodlnější tvorbu scén je uživateli umožněno načítat objekty ze souborů. Jedná se pouze o objekty fyzické (euklidovské i neeuklidovské). Načítání kamery a světel není podporováno. Geometrie musí být uložena v souboru formátu `.obj` a příslušný materiál ve formátu `.mtl`.

Načítat ze souboru lze i implementované neeuklidovské prvky. Je využito modifikovaného formátu `.obj`. Každý neeuklidovský prvek využívá jiných parametrů, jejichž popis a popis načítání lze najít v sekci 4.6.

4.3 Předzpracování dat

V této sekci jsou popsány úkony, jež je nutné provést před zahájením samotného vykreslování, neboli před začátkem hlavní smyčky programu. Průběh je znázorňen na obrázku 4.3. Předzpracování je rozděleno do dvou částí: příprava scény a příprava primárních paprsků. Tento přístup je výhodný pro snížení počtu nutných operací, které by se jinak musely dělat při vykreslování, což by snížilo výkon aplikace.



Obrázek 4.3: Na obrázku je znázorněna pre-processing pipeline, která se stará o přípravu dat potřebných pro vykreslování. Nejprve se připraví scéna transformací všech vrcholů a sestavením primitiv (trojúhelníků). Tyto akce imitují části OpenGL pipeline, konkrétně vertex shader a primitive assembly. Následně je nad každým objektem sestaveno obalové těleso pro akceleraci ray tracingu. Poslední akcí je výpočet směru všech primárních paprsků v souřadnicovém systému kamery vzhledem k požadovanému rozlišení a zornému poli.

Příprava scény

V rámci přípravy scény pro vykreslování jsou prováděny tři úkony. Jedním důvodem těchto operací je, že jsou potřebné pro vykreslení snímku, a tím druhým důvodem je další akcelerace vykreslování. Nejprve se provede transformace vrcholů, poté sestavení primitiv (trojúhelníků) a nakonec vytvoření obalového tělesa. Tyto akce jsou provedeny pro každý objekt ve scéně. Jelikož se jedná o proces, který se provede jen jednou za celý běh aplikace a jednotlivé operace nejsou výpočetně náročné, není pro ně použita žádná akcelerace v podobě paralelismu, jako tomu může být na grafických kartách, kde se typicky zpracovávají všechny vrcholy najednou.

Pro transformaci vrcholů ze souřadnicového systému objektu do souřadnicového systému scény nám stačí vynásobit každý vrchol objektu jeho transformační maticí. Při vykreslování se následně předpokládá, že všechny objekty, paprsky a jiné pomocné konstrukce byly transformovány a nachází se tedy v souřadnicovém systému scény. Obyčejně se pro transformaci vrcholů používá vertex shader na grafické kartě, ale naším cílem je veškeré výpočty provést na hlavním procesoru počítače.

Následuje sestavení primitiv, které se stejně jako transformace vrcholů provádí jako součást OpenGL pipeline. Výsledkem tohoto procesu jsou objekty reprezentující trojúhelníky, kde každý obsahuje tři ukazatele na jednotlivé vrcholy. Vytvořené trojúhelníky se následně používají zejména pro vykreslování, například pro hledání průsečíků paprsků s objekty.

Poslední částí přípravy scény je sestavení obalového tělesa nad objektem. Princip sestavení je jednoduchý. Jelikož jsou jako obalová tělesa použity osově orientované kvádry, stačí nám najít nejmenší a největší souřadnice v osách x, y, z ze všech vrcholů objektu. Obalové těleso je tedy reprezentováno dvěma body určující nejmenší a největší pozice vrcholů v jednotlivých osách.

Příprava primárních paprsků

Druhou částí přípravy dat před samotným vykreslováním je výpočet směru všech paprsků v souřadnicovém systému kamery. Tento výpočet není nutné dělat pro každý vykreslovaný snímek, jelikož kamera se ve svém souřadnicovém systému nepohybuje. Tím dosáhneme dalšího urychlení vykreslování.

Princip spočívá ve vytvoření imaginární průmětny, která leží v xy-rovině ve vzdálenosti jednoho bodu od počátku souřadnicového systému směrem do negativních hodnot osy z. V ose y se průmětna nachází v intervalu $\langle -1, 1\rangle$ a v ose x v intervalu $\langle -t, t\rangle$, kde t je poměr stran. Přepokládá se, že orientace generovaného snímku je na šířku, tedy $t > 1$. Směr paprsků v osách x, y, z vypočítáme následovně:

$$
\begin{aligned}
ray_x &= (2 \cdot \frac{x + 0.5}{width} - 1) \cdot aspectRatio \cdot \tan(\frac{\alpha}{2}),\\
ray_y &= 1 - 2 \cdot \frac{(y + 0.5)}{height} \cdot \tan(\frac{\alpha}{2}),\\
ray_z &= -1,
\end{aligned}
$$

kde x a y značí aktuální souřadnice pixelu v průmětně a α je úhel zorného pole. Takto předpřipravené paprsky slouží jako základ pro další vykreslování. Při vytváření obrazu jsou použity pro generování paprsků v souřadnicovém systému scény. Tento princip a konkrétní postup je dále rozebrán v sekci 4.5. Výsledné paprsky jsou uloženy sekvenčně v hlavní paměti pro podporu vektorizace.

4.4 Kamera a její pohyb ve scéně

Se způsobem reprezentace kamery jsme se seznámili v sekci 4.2. Zde se podíváme na konkrétní implementaci pohybu a otáčení kamery ve scéně na základě uživatelských vstupů. Způsob, jakým s kamerou manipulují neeuklidovské prvky, je vysvětlen v sekci 4.6. Dále je popsán i systém pro detekci kolizí kamery s objekty ve scéně.

Otáčení kamery

Při manipulaci s kamerou je její otáčení prvním řešeným problémem. Uživatel kamerou otáčí pomocí myši. Jako první je tedy zjištěna aktuální pozice kurzoru vzhledem k oknu aplikace. Tato informace je nadále využita pro výpočet vzdálenosti kurzoru od středu okna v obou osách a tyto hodnoty jsou použity jako parametr pro rotaci ve stupních. Při implementaci této funkcionality se vyskytly dva problémy. Tím jednodušším je omezení maximálního otočení kolem osy x. Jako maximální hodnota bylo experimentálně vybráno devadesát stupňů. Řešení je prosté, pokud by otočení v jakémkoliv směru limit přesáhlo, otočíme kameru jen o úhel chybějící do limitu. Druhým problémem bylo, že otáčení je prováděno kolem os v lokálním souřadnicovém systému. To v praxi znamená, že každá osa se mění na základě

otočení kolem os zbývajících. U otáčení kolem osy x nám to problém nezpůsobuje, nicméně pro osu y ano. Z tohoto důvodu bylo nutné ukládat aktuální rotaci kolem osy x tak, jak je popsáno v sekci 4.2. Správného otočení kolem osy y poté dosáhneme následovně:

1. Odstraníme aktuální otočení kolem osy x.
2. Provedeme rotaci kolem osy y.
3. Znovu aplikujeme otočení kolem osy x.

Pohyb kamery

U pohybu kamery musíme zařídit, aby pohyb nebyl závislý na snímcích za sekundu. Toho dosáhneme měřením uplynulého času od posledního zpracovávání pohybu. Pro měření času je využit časovač knihovny SFML, který se restartuje s každým přečtením. Výslednou vzdálenost, kterou kamera urazí, dostaneme vynásobením rychlostí kamery uplynulým časem v sekundách.

Směr pohybu je reprezentován jednotkovým vektorem v xz rovině globálního souřadnicového systému. Tento vektor se plní s využitím knihovny SFML, která umožňuje v reálném čase testovat stav kláves. Nevyužívají se tedy události systému. Pokud má dojít k pohybu (směrový vektor není nulový), je nejprve vyvolán systém pro detekci kolizí, který je popsán níže. Následně se může přejít k samotné translaci kamery. Narazíme na stejný problém jako při rotaci. Pokud je kamera otočená směrem dolů, směřuje tam i osa z. V lokálním souřadnicovém systému se tedy budeme pohybovat ve všech třech osách. Jednotlivé komponenty získáme následovně:

$$
\begin{aligned}
offset_x &= direction_x \cdot distance, \\
offset_y &= direction_z \cdot sin(\alpha) \cdot distance, \\
offset_z &= direction_z \cdot \cos(\alpha) \cdot distance,
\end{aligned}
$$

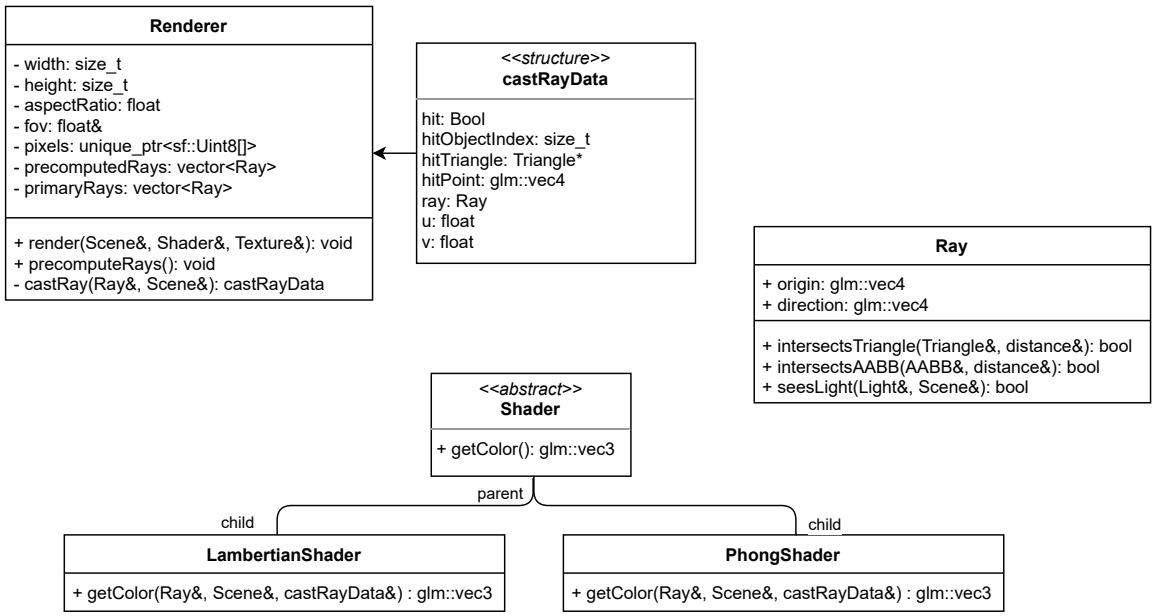
kde α je úhel představující aktuální otočení kolem osy x.

Systém detekce kolizí

Nejprve byl systém implementován tak, že po přesunutí kamery bylo otestováno, zda se kamera nenachází v obalovém tělese každého objektu. Pokud ano, přesunutí bylo vynulováno a kamera se opět nacházela na původním místě. Při implementaci neeuklidovských prvků, zejména portálů, se tento systém jevil jako velice nepraktický a z tohoto důvodu byl nahrazen jiným.

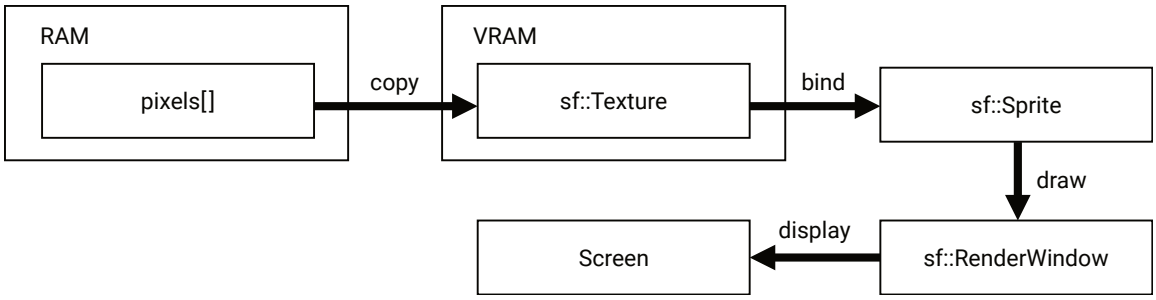
Nový a aktuálně implementovaný systém využívá preemptivní detekci kolizí. Ještě před samotným provedením pohybu je detekováno, zda při něm k nějaké kolizi dojde. Je využíváno sledování kolizního paprsku. Pokud má dojít k posunu kamery, je vytvořen kolizní paprsek s počátkem v aktuální pozici kamery a ve směru jejího pohybu. Následně je otestována srážka paprsku s každým objektem ve scéně. Pokud vzdálenost této kolize je menší než délka pohybu, nutně musí dojít ke střetnutí kamery s daným objektem. Pro zlepšení detekce je implementována rezerva pěti bodů vzdálenosti, což představuje minimální vzdálenost kamery od každého objektu. Pokud má skutečně dojít ke kolizi s euklidovským objektem, pohyb není uskutečněn.

4.5 Vykreslování



Obrázek 4.4: Diagram tříd demonstrující třídy související s vykreslováním scény.

Zde se podíváme na část aplikace zodpovědnou pro vykreslování. Část třídního diagramu je vidět na obrázku 4.4. Třída obsahuje jednorozměrné pole o délce $4 \cdot width \cdot height$, které imituje funkci framebufferu na grafické kartě. Každá čtveřice hodnot má velikost 32 bitů a představuje barvu jednoho pixelu ve formátu `RGBA`. Průhlednost pro vykreslování nemá žádný význam (po celou dobu běhu programu je nastavená na hodnotu 255, tedy žádná průhlednost), nicméně je nutná kvůli knihovně SFML, která se zde stará o samotné zobrazení vykresleného snímku na obrazovku uživatele. Toho dosáhneme tak, že výsledné pole zkopírujeme do struktury `sf::Texture`, která už se nachází přímo v paměti grafické karty. Tato textura je dále navázána na objekt `sf::Sprite`, který je umístěn přes celou plochu okna aplikace. Princip je demonstrován na obrázku 4.5.



Obrázek 4.5: Princip přenosu vykresleného snímku z operační paměti na obrazovku uživatele.

První nutnou akcí pro vykreslení snímku je transformace všech paprsků do globálního souřadnicovém systému. Jak bylo řečeno v sekci 4.3, máme v tuto chvíli již k dispozici předpočítané paprsky v souřadnicovém systému kamery. Převod provedeme vynásobením

počátku a směru každého paprsk transformační maticí kamery. Tento proces je paralelizován za použití OpenMP. Bylo vyzkoušeno a experimentálně zjištěno, že paralelizace přináší lepší výkon než SIMD, a to jak s nativní podporou, tak s explicitním příkazem pro SIMD skrz OpenMP.

Jakmile máme připraveny paprsky, můžeme začít s iterací přes každý pixel a vysílat jednotlivé paprsky do scény. K tomu byla implementována samostatná metoda společně se strukturou, která obsahuje veškerá data potřebná pro další postup. Algoritmus funkce ve zjednodušené podobě je demonstrován v algoritmu 1. Informace obsažené ve struktuře lze vidět v třídním diagramu 4.4.

Pro výpočet průsečíku paprsku s trojúhelníkem je použit algoritmus Möller-Trumbore, který byl představen v teoretické části v sekci 2.2. Konkrétní implementace byla převzata z nástroje GPUEngine[5] a lehce upravena pro potřeby aplikace. Hlavní změny se týkaly použití matematických funkcí knihovny GLM. Tyto funkce měly oproti explicitnímu výpočtu značně horší výkon. Další změnou bylo, že funkce nyní vrací i barycentrické koordináty průsečíku, které se dále používají pro texturování. K výpočtu průsečíku paprsku s obalovým tělesem (AABB) byl zvolen algoritmus navržený Tavianem Barnesem v internetovém článku[6], což je optimalizovaná verze slab metody [15]. Jedná se o algoritmus bez jakéhokoliv větvení a s minimálním počtem operací dělení.

Algorithm 1: Pseudokód metody pro vysílání paprsků do scény

```
1  castRayData data;
2  foreach object in scene do
3      if ray intersects AABB then
4          foreach triangle in object do
5              if ray intersects triangle then
6                  if distance < minDistance then
7                      minDistance = distance;
8                      Update data;
9                  end
10             end
11         end
12     end
13 end
```

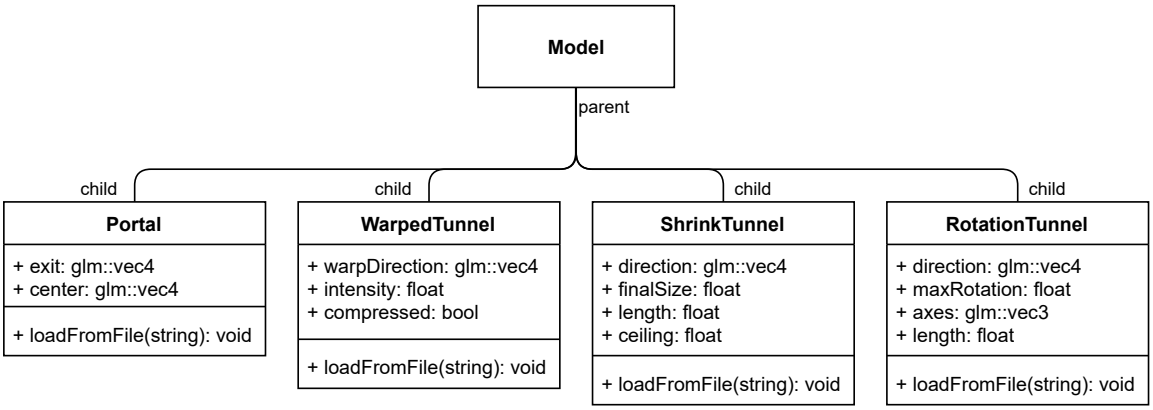
Po skončení funkce tedy máme k dispozici naplněnou strukturu s daty, podle kterých je určen další průběh vykreslování. Pokud došlo ke kolizi paprsku s objektem, je vyvolán systém osvětlování, který rozhodne o výsledné barvě. Pokud ke kolizi naopak nedošlo, je výsledná barva zjištěna z textury aplikované na skybox. Jako poslední krok vykreslování je aktualizace barvy v poli pixelů a zkopírování celého pole do použité textury uložené na grafické kartě.

[5] GPUEngine je knihovna pro 3D vykreslování vyvíjená na fakultě FIT. Repozitář knihovny: https://github.com/Rendering-FIT/GPUEngine.

[6] https://tavianator.com/2011/ray_box.html

4.6 Neeuklidovské prvky

Každý neeuklidovský prvek je implementován ve vlastní třídě, která dědí z třídy popisující fyzický objekt scény, jak je znázorněno na třídním diagramu 4.6. Znamená to, že každý prvek se skládá ze sítě trojúhelníků a může nabývat libovolného tvaru. Informace o texturování a materiálu nejsou využity, jelikož jsou prvky buď průhledné (tunely), nebo v případě portálů zobrazují část scény z jiného pohledu.



Obrázek 4.6: Diagram tříd demonstrující implementované neeuklidovské prvky.

Při tvorbě aplikace byl nejprve vytvořen vykreslovací systém s pohybem kamery tak, jak je popsáno v sekcích 4.5 a 4.4. Tvorba neeuklidovských prvků tedy představuje jejich začlenění do již existujícího systému. V případě prvků, které ovlivňují vykreslování, je nutné určitým způsobem závislým na konkrétním prvku manipulovat s primárními paprsky. Nicméně je třeba rozšířit i systém pro pohyb kamery, abychom dosáhli synchronizace pohybu s vykreslováním. Bez toho by všechny prvky působily nepřesvědčivě.

Bylo tedy nutné rozšířit metodu pro vysílání paprsků do scény. Základ se nijak nemění, nejprve nalezneme průsečík s nejbližším objektem tak, jak je ukázáno v algoritmu 1. Poté ale musíme zkontrolovat, zda paprsek nenarazil do nějakého neeuklidovského prvku. Pokud ano, vytvoříme nový, upravený paprsek, který znovu vyšleme do scény. Metoda se tak stává rekurzivní.

Pro synchronizaci pohybu s prvky je nutné upravit směr a rychlost pohybu kamery v závislosti na konkrétním prvku. Zde mohou nastat tři možnosti. První je, že do nějakého neeuklidovského prvku vstupujeme. To zjistíme pomocí preemptivní detekce kolizí. V takovém případě musíme nejprve přistoupit těsně k objektu standardním způsobem a poté provést pohyb uvnitř. Druhá možnost je obdobou první, jen z objektu vystupujeme ven. Poslední situací je, že se kamera nachází uvnitř nějakého objektu již při počátku pohybu. Nejprve upravíme charakteristiku prováděného pohybu, až poté provedeme detekci kolizí.

Portály

Portály byly prvním implementovaným neeuklidovským prvkem. Umožňují přesun kamery a paprsků na libovolnou vzdálenost. Jsou implementovány jako jednosměrné, pokud portálem projdeme, nemůžeme se stejnou cestou vrátit zpět. Můžeme do nich však vejít z obou stran. Další limitací je jejich orientace. Vstup a výstup je vždy orientován stejným směrem. To je patrné z obrázku 4.7. Portály také mohou být samostatnými jednotkami v prostoru, nemusí být například nalepené na existující stěnu. Kromě vrcholů s indexováním je portál

reprezentován bodem v prostoru, který značí místo destinace. Dále také geometrickým středem, který slouží k výpočtu vzdálenosti pro přesun.

Pro načtení portálu ze souboru používající modifikovaný formát `.obj` je nutné, aby soubor obsahoval následující položky:

```
c <x> <y> <z>
e <x> <y> <z>
```

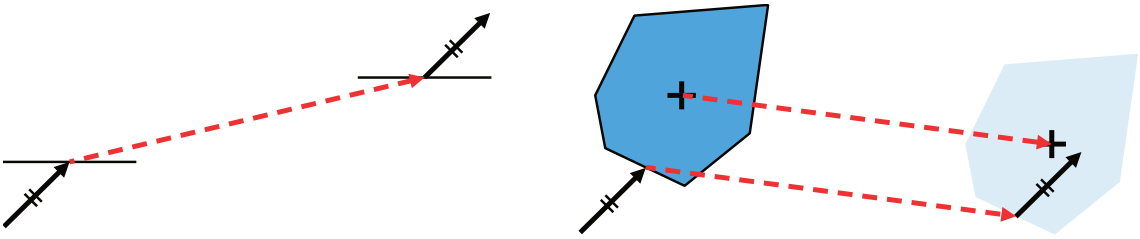
Položka `c` značí geometrický střed portálu a `e` destinaci. `x`, `y` a `z` jsou desetinná čísla reprezentující souřadnice daných bodů.

Při vykreslování dosáhneme žádaného efektu přesunem paprsků, které zasáhnou portál. Při kolizi je vytvořen nový paprsek, jehož počátek bude v bodě:

$$origin = H + E - C,$$

kde H je bod kolize paprsku s portálem, E je bod reprezentující destinaci portálu a C je jeho geometrický střed.

Implementace průchodu kamery portálem využívá stejné rovnice, nicméně je nutné provést pár úprav. Pokud víme, že kamera při svém pohybu narazí do portálu, víme i vzdálenost, ve které náraz nastane. Tu použijeme pro výpočet zbývající vzdálenosti, kterou kamera musí urazit po přemístění. Nejprve tedy dojde k teleportaci kamery na místo určení a poté je ještě posunuta o zbývající vzdálenost ve stejném směru.



Obrázek 4.7: Ukázka, jak portály interagují s paprskem vrženým do scény. V levé části je vidět použití 2D portálů. Naopak pravá část princip ukazuje na 3D portálu s přenosem vzdálenosti od jeho středu.

Zakřivené tunely

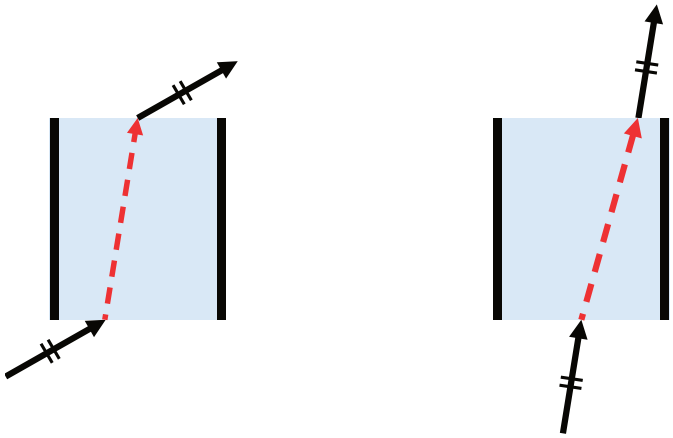
Zakřivené tunely modifikují paprsky buď jejich protažením, nebo zkrácením v určitém směru. Vyvolávají tak efekt, že je prostor v nich buď stlačený, nebo protažený. Taktéž mohou mít libovolný tvar složený z trojúhelníků, nicméně pro nejlepší efekt je doporučeno tunel vytvořit v podobě hranatého tunelu a obalit jej do euklidovských stěn. Princip průchodu tunelem je demonstrován na obrázku 4.8. Parametry tunelu jsou navíc jeho směr, intenzita zakřivení a binární hodnota označující, jestli tunel prostor stlačuje, nebo naopak protahuje. Soubor s popisem tunelu tak musí obsahovat následující:

```
c <binary_value>
d <x> <y> <z>
i <value>
```

`c` je parametr s binární hodnotou, `d` je směr tunelu se souřadnicemi `x`, `y`, `z` a `i` značí intenzitu tunelu desetinným číslem. U stlačeného tunelu se intenzita pohybuje v intervalu $\langle 1, 8 \rangle$. U hodnot menších než 1 nedochází ke zkrácení a u hodnot větších než 8 sice k většímu zkrácení dojde, ale začnou se projevovat problémy se synchronizací kamery, zejména při nízkém počtu snímků za sekundu. U tunelu protaženého je použit interval $\langle 0, 1 \rangle$, kde 0 znamená žádnou změnu a 1 nekonečný tunel. Směr paprsku po vstupu do stlačeného tunelu můžeme vypočítat následovně:

$$direction = \vec{r} + \left(\vec{d} \cdot \vec{r}\right) \cdot \vec{d} \cdot i,$$

kde $\vec{r}$ je směr původního paprsku, $\vec{d}$ je směr tunelu a i je jeho intenzita. Výpočet směru pro protažený tunel se upraví pouze změnou znaménka plus na mínus. Při výstupu paprsku z tunelu nastavíme jeho směr na původní směr paprsku, který do tunelu vstupoval.



Obrázek 4.8: Demonstrace změny směru paprsku při průchodu zakřiveným tunelem. Tunely jsou implementovány ve dvou verzích. Tunel se stlačeným prostorem je v levé části obrázku a tunel s prostorem roztaženým je v části pravé.

Tento tunel přinesl ještě jeden problém při implementaci. Pokud se kamera nachází uvnitř tunelu, musíme generovat již zakřivené paprsky. Toto je prováděno hned po transformaci paprsků do globálního souřadnicového systému, která je popsána v sekci 4.5.

Synchronizace kamery probíhá ve dvou krocích. Pokud má dojít ke vstupu do tunelu, je pozice kamery nastavena na místo vstupu a přesahující vzdálenost je zanedbána. Tento přístup není sice úplně přesný, ale rozdíl je nepostřehnutelný. Ve chvíli, kdy je zjištěno, že se kamera nachází uvnitř tunelu, je směr pohybu upraven stejným způsobem jako směr paprsku s jedním rozdílem. Zakřivení proběhne pouze v rovině xz. Pro dosažení efektu, že cesta vně tunelu je pomalejší než cesta skrz (u stlačeného tunelu), se musí upravit i rychlost pohybu. V tomto případě toho není dosaženo změnou rychlosti kamery, ale změnou vzdálenosti, kterou má kamera urazit. Novou vzdálenost ve stlačeném tunelu získáme takto:

$$distance = \left(v \cdot |\vec{d} \cdot \vec{m}| \cdot i\right) + \left(v \cdot \left(1 - |\vec{d} \cdot \vec{m}|\right)\right)$$

a v protaženém tunelu takto:

$$distance = \left(v \cdot |\vec{d} \cdot \vec{m}| \cdot (1 - i)\right) + \left(v \cdot \left(1 - |\vec{d} \cdot \vec{m}|\right)\right),$$

kde v je původní vzdálenost, $\vec{m}$ je směr pohybu, $\vec{d}$ je směr zakřivení tunelu a i je jeho intenzita. Poslední věcí, kterou je nutno změnit, je směr kolizního paprsku. Jeho směr nastavíme na nový směr pohybu kamery.

Škálovací tunely

Škálovací tunel zvětšuje nebo zmenšuje uživatele v závislosti na směru, ve kterém byl tunel procestován. Opět je nejlepší variantou hranatý tvar obalený standardními stěnami. Parametry jsou zde směr tunelu, výsledná procentuální velikost při zmenšení, délka tunelu v jeho směru a souřadnice v ose y označující výšku stropu v tunelu. Soubor s jeho popisem musí tyto položky obsahovat:

```
d <x> <y> <z>
i <value>
l <value>
c <value>
```

kde jednotlivé prvky jsou v pořadí jejich popisu.

Škálovací tunel nijak neovlivňuje vykreslování, pouze manipuluje s kamerou. Změna velikosti uživatele se provádí pouze při pohybu v, nebo proti směru tunelu. V rámci aplikace je pevně stanovená počáteční souřadnice y kamery i podlahy. Zmenšení se tak provádí relativně vůči rozdílu těchto dvou hodnot. Hlavním problémem je výpočet posunu kamery v ose y. K tomu si musíme uchovat výšku kamery při vstupu do tunelu. Posun získáme následovně:

$$distance_y = (h_c - h_s) \cdot \frac{v \cdot \vec{m} \cdot \vec{d}}{l} \cdot (1 - i)\,,$$

kde h_c je výška kamery při vstupu, h_s je výška podlahy, v je celková vzdálenost pohybu, $\vec{m}$ je jeho směr, $\vec{d}$ je směr tunelu, l jeho délka a i je výsledná velikost. V tuto chvíli musíme vyřešit kolizi kamery se stropem při zvětšování. Pokud se po provedení posunutí v ose y nebude kamera nacházet nad stropem, nedošlo ke kolizi a můžeme pohyb provést. Nakonec stejným způsobem upravíme i rychlost kamery.

Rotační tunely

Rotační tunel je velice podobný tomu škálovacímu. Nijak neovlivňuje vykreslování, pouze manipulaci s kamerou. Parametry tohoto prvku jsou opět směr, maximální rotace ve stupních (celková rotace při procestování celého tunelu), tři binární hodnoty označující osy, kolem kterých bude prováděna rotace a délka tunelu. Relevantní část `.obj` souboru vypadá takto:

```
d <x> <y> <z>
i <value>
a <bin_x> <bin_y> <bin_z>
l <value>
```

prvky `d` a `l` značí směr a délku tunelu, `i` je maximální rotace a `a` určuje osy pro rotaci. Například pro hodnoty $(0, 1, 1)$ bude tunel provádět rotaci kolem osy y a osy z zároveň.

Při každé instanci pohybu je vypočítán zlomek uražené vzdálenosti vůči celkové délce tunelu. Tento zlomek vynásobený maximální rotací tunelu udává, o kolik stupňů musíme s kamerou otočit. Důležitá je opět úprava směru kolizního paprsku.

4.7 Osvětlení a texturování

V momentě, kdy víme, který bod objektu bude zobrazen v daném pixelu, můžeme začít řešit jeho barvu. V projektu jsou implementovány celkem dva druhy osvětlení: Lambertovo a Phongovo. Jejich rozdělení do tříd je znázorněno v diagramu 4.4. Vytvoření abstraktní třídy a následné dědění různých oslvětlovacích modelů nám umožňí dvě věci. Můžeme kdykoliv přijít a implementovat další osvětlovací model bez zásahu do existujícího kódu. Druhou výhodou je, že použitý osvětlovací model se dá měnit za běhu programu. Stisknutím příslušeného tlačítka na klávesnici se změní ukazatel na abstraktní třídu. Tento ukazatel je v hlavní smyčce programu posílán jako argument do funkce pro vykreslení nového snímku, kde je při kolizi paprsku s objektem volána jeho předefinovaná funkce pro výpočet barvy.

Jako první je nutné vypočítat normálu povrchu. Toho dosáhneme vektorovým součinem dvou stran trojúhelníku zasaženého paprskem. Aplikace předpokládá pořadí vrcholů proti směru hodinových ručiček. Následuje zjištění výsledné barvy objektu. Pokud má objekt aplikovanou texturu, zjistí se barva pomocí interpolace texturovacích souřadnic barycentrickými koordináty. V případě absence textury obsahuje třída popisující materiál položku, která určuje barvu objektu. Každý komponent barvy je normalizován v intervalu $\langle 0, 1 \rangle$.

Nyní můžeme přistoupit k výpočtu samotné barvy, který se může lišit v závislosti na použitém osvětlovacím modelu. V Lambertově osvětlovacím modelu získáme barvu následovně:

$$color = \sum_{m \in lights} \left(v_m \cdot albedo \cdot i_m \cdot \max\left(0, \vec{N} \cdot \vec{L}\right)\right),$$

kde $\vec{N}$ je normála povrchu, $\vec{L}$ je směr ke světlu. v_m nabývá hodnot $\{0, 1\}$ a určuje, jestli se bod nachází ve stínu vůči danému světlu. Hodnotu zjistíme vysláním stínového paprsku z daného bodu směrem ke světlu. Pokud paprsek protne jakýkoliv objekt scény ve vzdálenosti kratší, než je vzdálenost ke světlu, dostaneme hodnotu 0. Hodnota 1 značí, že světlo je z daného bodu viditelné. Člen i_m představuje množství světla. U distančních světel jej získáme vynásobením barvy světla jeho intenzitou. U světel bodových upravíme výpočet takto:

$$i_m = \frac{intensity \cdot color}{4\pi \cdot r^2},$$

kde r je vzdálenost ke světlu.

U Phongova osvětlení je situace o něco složitější. Jak je popsáno v kapitole 2.4, je osvětlení složeno ze tří složek. Difuzní složku vypočítáme stejně jako barvu při použití Lambertova osvětlení. Spekulární složku získáme následovně:

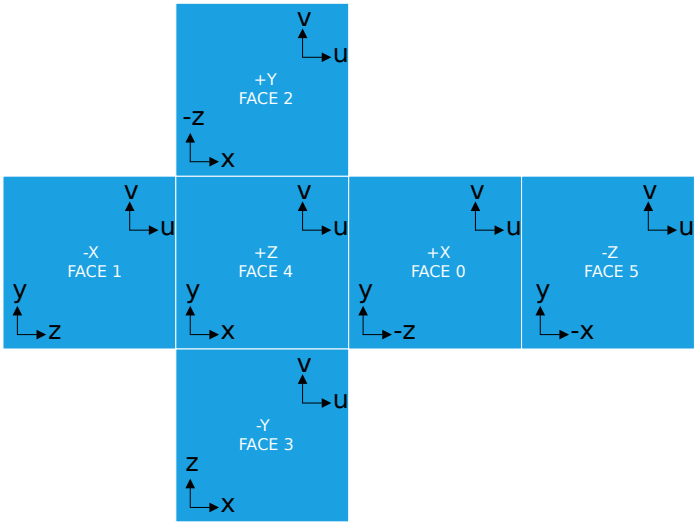
$$specular = \sum_{m \in lights} \left(v_m \cdot i_m \cdot \max\left(0, \vec{R} \cdot \vec{D}\right)^n\right),$$

$\vec{R}$ je vektor odrazu světla od povrchu a n je míra lesklosti. Ambientní složka je konstantní v celé scéně a pro celou aplikaci nastavená na malou hodnotu bílého světla. Výslednou barvu získáme kombinací všech složek takto:

$$color = diffuse * k_d + specular * k_s + ambient * k_a$$

4.8 Skybox

Pro zvětšení hloubky obrazu a lepšího vizuálního dojmu ze scény je v aplikaci implementován skybox. Jako forma bylo zvoleno obalení scény do krychle, na kterou je namapována textura. Ukázka, jak je konkrétně řešeno mapování a indexování stran, je vidět na obrázku 4.9. Při vytváření objektu je nutné předat aplikaci informaci o názvu souboru s texturou. Pro načítání textury je využita knihovna SFML, která podporuje řadu často používaných formátů bitmapových souborů. Program očekává, že poměr stran vstupního obrázku bude 4 : 3. V případě, že poměr bude jiný, nastane neočekáváné chování. Po načtení je vytvořeno mapování části textury na určitou stranu krychle tak, že ke každé straně jsou uloženy souřadnice levého dolního rohu.



Obrázek 4.9: Ukázka systému mapování textury na krychli v aplikaci, včetně indexování jednotlivých stran a směru u, v souřadnic. Při výchozí pozici kamery se uživatel dívá na stranu s číslem čtyři.[7]

Vytažení barvy z textury se provádí při vykreslování, pokud paprsek minul všechny objekty ve scéně. Funkcionalita je implementována v metodě, která provádí mapování směru paprsku. Vektor (x, y, z) je nejprve převeden na texturovací souřadnice (u, v), které jsou v intervalu $\langle 0, 1 \rangle$ a následně na souřadnice pixelu v souboru s texturou (x, y). Z toho je patrné, že výsledná barva nijak nezávisí na pozici kamery, pouze na směru pohledu. To způsobí, že skybox je statický v rámci celé scény. Implementace převodu byla převzata a upravena pro potřeby aplikace.[7]

[7] Kód i obrázek byl převzat z: https://en.wikipedia.org/wiki/Cube_mapping.

Kapitola 5

Závěr

Cílem této práce bylo navrhnout a implementovat interaktivní aplikaci demonstrující vykreslování scény obohacené o neeuklidovské prvky metodou sledování paprsků v reálném čase. Nejprve byly rozebrány a popsány obecné principy relevantní k práci, zejména týkající se metody sledování paprsků a neeuklidovské geometrie jak v matematice, tak i v běžném světě. Teoretické poznatky vedly k vytvoření návrhu aplikace, která demonstruje vybrané neeuklidovské prvky v několika scénách.

Výsledkem je implementovaná aplikace, která pro veškeré výpočty využívá CPU počítače. Sledování paprsků je akcelerováno zejména předzpracováním potřebných dat, paralelizací výpočtů za pomocí knihovny OpenMP a obalovými tělesy. Jsou implementovány celkem čtyři druhy neeuklidovských prvků. Prvním jsou portály, které přemisťují jak paprsky při vykreslování, tak i kameru. Dále existují zakřivené tunely, jež mění topologii prostoru uvnitř. Třetím a čtvrtým druhem jsou rotační a škálovací tunely, které nijak neovlivní vykreslování, pouze manipulaci s kamerou při průchodu. Výsledná aplikace obsahuje celkem osm minimalistických ukázkových scén. V rámci práce bylo vytvořeno krátké video prezentující práci, které je uloženo na přiloženém datovém médiu.

Projekt má hned několik možností na další rozšíření. Tím nejzákladnějším by byl převod celého vykreslování na grafickou kartu počítače. To by zajistilo podstatně lepší výkon a umožňilo tvorbu komplexních scén kombinující různé neeuklidovské prvky. Dalším rozšířením by mohlo být přidání více druhů neeuklidovských prvků, zejména pak takových, které využívají striktně matematického významu neeuklidovské geometrie. Poslední možností by mohla být tvorba logické počítačové hry.

Literatura

[1] Appel, A. Some Techniques for Shading Machine Renderings of Solids. In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. New York, NY, USA: Association for Computing Machinery, 1968, s. 37–45. AFIPS '68 (Spring). DOI: 10.1145/1468075.1468082. ISBN 9781450378970. Dostupné z: https://doi.org/10.1145/1468075.1468082.

[2] Dubla, P. *Interactive Global Illumination on the CPU*. Disertační práce.

[3] Hapala, M. a Havran, V. Review: Kd-tree Traversal Algorithms for Ray Tracing. *Computer Graphics Forum*. 2011, sv. 30, č. 1, s. 199–213. DOI: 10.1111/j.1467-8659.2010.01844.x. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2010.01844.x.

[4] Hlavatý, V. *Úvod do neeuklidovské geometrie*. Jednota Československych matematiku a fysiku, 1949. Edice Kruh.

[5] Hu, Y., Wang, W., Li, D., Zeng, Q. a Hu, Y. Parallel BVH Construction Using Locally-Density Clustering. *IEEE Access*. Červenec 2019, PP, s. 1–1. DOI: 10.1109/ACCESS.2019.2932151.

[6] Kay, T. L. a Kajiya, J. T. Ray Tracing Complex Scenes. New York, NY, USA: Association for Computing Machinery. srpen 1986, sv. 20, č. 4, s. 269–278. DOI: 10.1145/15886.15916. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/15886.15916.

[7] Kutuzov, B. *Lobačevského geometrie a elementy základů geometrie*. ČSAV, 1953. Sekce matematicko-fysikální.

[8] McAuley, S., Hill, S., Martinez, A., Villemin, R., Pettineo, M. et al. Physically Based Shading in Theory and Practice. In: *ACM SIGGRAPH 2013 Courses*. New York, NY, USA: Association for Computing Machinery, 2013. SIGGRAPH '13. DOI: 10.1145/2504435.2504457. ISBN 9781450323390. Dostupné z: https://doi.org/10.1145/2504435.2504457.

[9] Meagher, D. Geometric Modeling Using Octree-Encoding. *Computer Graphics and Image Processing*. Červen 1982, sv. 19, s. 129–147. DOI: 10.1016/0146-664X(82)90104-6.

[10] Mei, G. *RealModel-a system for modeling and visualizing sedimentary rocks*. Disertační práce.

[11] Möller, T. a Trumbore, B. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphics Tools*. Srpen 2005, sv. 2. DOI: 10.1145/1198555.1198746.

[12] Pavlíček, J. B. *Základy neeuklidovské geometrie Lobačovského*. Přírodovědecké nakladatelství, 1953.

[13] Pharr, M., Jakob, W. a Humphreys, G. *Physically Based Rendering: From Theory to Implementation*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016. ISBN 0128006455.

[14] Phong, B. T. Illumination for Computer Generated Pictures. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. červen 1975, sv. 18, č. 6, s. 311–317. DOI: 10.1145/360825.360839. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/360825.360839.

[15] Sefi, S. Ray Tracing Tools for High Frequency Electromagnetics Simulations. Leden 2003.

[16] Whitted, T. An Improved Illumination Model for Shaded Display. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. červen 1980, sv. 23, č. 6, s. 343–349. DOI: 10.1145/358876.358882. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/358876.358882.